Open-Source ANSS Quake Monitoring System Software

J. Renate Hartog*[1], Paul A. Friberg[2], Victor C. Kress[1], Paul Bodin[1], and Rayomand Bhadha[3]

Abstract

ANSS stands for the Advanced National Seismic System of the U.S.A., and ANSS Quake Monitoring System (AQMS) is the earthquake management system (EMS) that most of its member regional seismic networks (RSNs) use. AQMS is based on Earthworm, but instead of storing files on disk, it uses a relational database with replication capability to store pick, amplitude, waveform, and event parameters. The replicated database and other features of AQMS make it a fully redundant system. A graphical user interface written in Java, Jiggle, is used to review automatically generated picks and event solutions, relocate events, and recalculate magnitudes. Add-on mechanisms to produce various post-earthquake products such as ShakeMaps and focal mechanisms are available as well. It provides a configurable automatic alarming and notification system. The Pacific Northwest Seismic Network, one of the Tier 1 ANSS RSNs, has modified AQMS to be compatible with a freely available, capable, open-source database system, PostgreSQL, and is running this version successfully in production. The AQMS Software Working Group has moved the software from a subversion repository server hosted at the California Institute of Technology to a public repository at gitlab.com. The drawback of AQMS as a whole is that it is complex to fully configure and comprehend. Nevertheless, the fact that it is very capable, documented, and now free to use, might make it an attractive EMS choice for many seismic networks.

Cite this article as Renate Hartog, J., P. A. Friberg, V. C. Kress, P. Bodin, and R. Bhadha (2019). Open-Source ANSS Quake Monitoring System Software, *Seismol. Res. Lett.* **91**, 677–686, doi: 10.1785/0220190219.

Introduction

Regional seismic networks (RSNs) need software to automatically, and in semi-real time, process continuously streaming waveform data to identify seismic signals, detect events, and determine their locations, magnitudes, and if possible, focal mechanisms. At most RSNs, automatically produced event parameters are reviewed and manually adjusted by analysts, requiring a graphical user interface (GUI). Lastly, event parameters are stored and as such comprise the RSNs earthquake catalog, which requires a storage mechanism. These elements together are known as an earthquake management system (EMS).

Commercial EMS solutions are less attractive for long-term, publicly funded, seismic monitoring networks due to cost and the risk that a product is discontinued. As a result, operations at many RSNs are based on the open-source real-time processing Earthworm package, which was developed in the early 1990s by the U.S. Geological Survey (USGS) and is still actively maintained (Johnson *et al.*, 1995; Olivieri and Clinton, 2012; also see Data and Resources). However, Earthworm itself does not provide a GUI to allow for analyst review. There are a variety of custom software modules that can be used with Earthworm to allow seismologists to re-pick phases and do further analysis of events. One popular analysis program is SEISAN (Havskov and Ottemöller, 1999). The Istituto Nazionale di Geofisica e Vulcanologia National Earthquake Center developed a MySQL database (moledb) to store Earthworm messages and a web interface (moleface) that allows browsing of the stored picks and other parameters (Quintiliani and Pintore, 2013). Another popular EMS is SeisComP3, which has a full suite of GUIs for postevent analysis (e.g., Hanka *et al.*, 2010; Data and Resources).

Olivieri and Clinton (2012) provided a comprehensive review and comparison of both SeisComP3 and Earthworm to help network operators select an EMS. They identified the lack of a central database, a postevent review and analysis tool, and a mechanism to trigger automatic event notifications as downsides of Earthworm and the robust time-tested algorithms, documentation, and active user and developer community as its strengths. They mentioned the ANSS Quake Monitoring System (AQMS) as an interactive

1. Pacific Northwest Seismic Network, Department of Earth and Space Sciences, University of Washington, Seattle, Washington, U.S.A.; 2. Instrumental Software Technologies, Inc., Saratoga Springs, New York, U.S.A.; 3. Southern California Seismic Network, Seismological Laboratory, California Institute of Technology, Pasadena, California, U.S.A.

*Corresponding author: jrhartog@uw.edu

Figure 1. Authoritative regions (light gray) of public networks that use Advanced National Seismic System (ANSS) Quake Monitoring System (AQMS) for processing earthquakes, retrieved from the Geoserve website (see Data and Resources) on 9 August 2019. Alaska Volcano Observatory (AVO; AV, authoritative near volcanoes, the Alaska Earthquake Information Center, AEIC; AK, uses Antelope instead of AQMS), Center for Earthquake Research and Engineering (CERI; ET, MN), Hawaii Volcano Observatory (HVO; HV), Lamont–Doherty Seismic Network (LDSN; LD), Northern California Seismic System (NCSS; NC, BK, BP, BG), Pacific Northwest Seismic Network (PNSN; UW, UO, CC), Puerto Rico Seismic Network (PRSN; PR), Southern California Seismic Network (SCSN; CI, AZ, ZY), and University of Utah Seismic Stations (UUSS; UU, WY). Not shown: Centro de Investigación Científica y de Educación Superior de Ensenada Baja, California, which also uses AQMS (CICESE; BC, LP, RB). The color version of this figure is available only in the electronic edition.

Earthworm-based EMS that does provide a central database, postevent review, and analysis tools and automatic event notifications, but at the time of their review AQMS relied on the commercially licensed Oracle database system and was not yet an open release and therefore not further discussed.

AQMS was developed in the 2000s and is largely based on the TriNet system that was developed at Caltech. It then expanded northward to the Northern California Seismic System and became the California Integrated Seismic Network (CISN) TriNet system. In 2008, ANSS management at the USGS decided that all the Tier 1 ANSS networks should adopt the CISN system at which point it was renamed to AQMS (Friberg *et al.*, 2010). The adoption of AQMS at the various regional networks took tremendous effort, due to long-standing custom solutions that had to be replaced and the need to incorporate changes made to AQMS during the same period. Nevertheless, by 2013 seven USGS-funded networks were running AQMS as their production systems. Figure 1 shows the geographic distribution and sizes of networks that are successfully using AQMS. AQMS has been deployed on computers with Linux operating system (OS) and Solaris OS in the past. Despite the fact that Earthworm runs on the Windows OS, AQMS is not supported on Windows.

Until recently, AQMS required the use of an Oracle relational database system, which made it prohibitively expensive to run for many networks. The Pacific Northwest Seismic Network (PNSN), one of the Tier 1 ANSS RSNs, faced difficulties with the license cost, and we modified the software to be compatible with the freely available, capable, open-source database system, PostgreSQL. The port was completed at the end of 2018, and PNSN has been running their production AQMSs using PostgreSQL on Linux since 1 October 2018. The AQMS Software Working Group is moving the AQMS software and documentation from a subversion repository

server hosted at the California Institute of Technology to a public repository at gitlab.com with the goal of making the system more accessible to other seismic networks.

Earthworm at the Base

AQMS uses the Earthworm (Johnson *et al.*, 1995; Olivieri and Clinton, 2012) automatic processing system. Written in the C programming language, Earthworm provides multiple waveform processing modules, multiple phase pickers (e.g., *pick_ew, pick_FP, coda_picker, carlstatrig*), a ground-motion parameters module (*gmew*), a local magnitude algorithm (*localmag*), one subnet coincidence trigger (*carlsubtrig*) event detector, and one event detector and pick associator named *binder_ew* (Johnson *et al.*, 1997). To obtain refined hypocenter locations, Earthworm can be configured to run the Hypoinverse (Klein, 2002) or NonLinLoc (Lomax *et al.*, 2000, 2009) earthquake locators. Earthworm achieves its modularity and efficiency by passing compact messages of different types into circular shared memory buffers, so-called rings. Typically, an Earthworm module subscribes to a ring to receive one type of message and publishes to a ring its own output message to be picked up by the next module in the processing chain. Earthworm modules obtain all the information they need, such as channel files and parameter settings, from configuration files. Tuning Earthworm to work well for a network is an art as well as a science. A full description of Earthworm is beyond the scope of this article, and we refer the reader to Earthworm's documentation webpage instead (see Data and Resources).

Interserver transmission is handled by import-export client-server Transmission Control Protocol (TCP) protocol or User Datagram Protocol (UDP) multicast-broadcast modules that allow messages to be spread across multiple cooperating Earthworm instances. These currently available methods of shipping messages between computers lack flexibility (one-to-one TCP-based modules) and/or robustness (the UDP-based modules). We imagine that at some point in the future, Earthworm messages might be exchanged using modern publish-subscribe Message Brokers such as Apache Kafka or RabbitMQ, to be able to run pickers and associators on different physical or virtual systems without fear of adding significant latency or dropping messages.

The Database at the Core

The database at the core of AQMS fulfills several functions. It stores the earthquake catalog, phase picks, amplitude measurements, coda duration measurements, and so on. However, it also holds the metadata for the seismic channels used now, and in the past, including calculated parameters such as station magnitude corrections. For the PostgreSQL version, a Python script, *loadStationXML*, is available to load International Federation of Digital Seismograph Networks (FDSN) StationXML into the database (see Data and Resources). Most RSNs that use Oracle have tools to load dataless Standard for the Exchange of Earthquake Data (SEED) into the database. The AQMS database schema includes a section that can store hardware inventory; however, not many of the RSNs use that part of the schema. Instead, the ANSS supports an ANSS Station Information System (SIS; Yu *et al.*, 2017) hosted by Caltech that the RSNs are encouraged to use. SIS can publish both FDSN StationXML and dataless SEED metadata files. All the AQMS modules query database tables upon startup to determine which channels they should process.

The AQMS database schema can store the complete history of the detection and analysis of seismic events. All automatic and reviewed measurements as well as all hypocenter and magnitude estimates are saved to the database by the AQMS software stack. This allows an RSN to periodically review the overall system performance; for example, one can compare final estimates of location and magnitude to the automatic estimates. The Oracle version of the schema can be perused at NCEDC database project (see Data and Resources). The PostgreSQL schema is almost identical but differs in details such as data types and stored function implementations (see Data and Resources).

The database contains many stored functions to enforce logic rules and orchestrate external processes, as well as provide several data integrity constraints to validate values of specific fields in some of the tables. To port the AQMS software to PostgreSQL, we translated all the necessary Procedural Language for Structured Query Language (PL/SQL) Oracle-stored procedure packages into PostgreSQL Procedural Language for PostgreSQL-flavored Structured Query Language (PL/pgSQL) functions. To ensure that client software, such as the real-time programs described in the next section and the GUI Jiggle, would be able to connect to either an Oracle AQMS database or a PostgreSQL AQMS database, we kept the call signature of stored functions the same, even if the internal implementation had to be different.

We wrote one custom function for PostgreSQL in C to be able to serve miniSEED files from the database server hardware via the database server to a remote client (Jiggle). The Oracle AQMS database achieves this via stored Java code instead. A stored PL/SQL (Oracle) or PL/pgSQL (PostgreSQL) function wraps around calls to these custom database extensions to provide an identical Application Programming Interface (API).

AQMS Automatic Processing

Event parameters

Figure 2 illustrates the data flow in a minimal AQMS system. It uses Earthworm to generate subnet coincident triggers via *carlstatrig-carlsubtrig* (see Data and Resources) as well as more selective *binder* events (Johnson *et al.*, 1997) that get funneled through the earthworm mega-module (sometimes referred to as "the sausage") into Hypoinverse (Klein, 2002). Earthworm can be configured with many different rings. The example system in Figure 2 has a ring for passing Tracebuf2 waveform

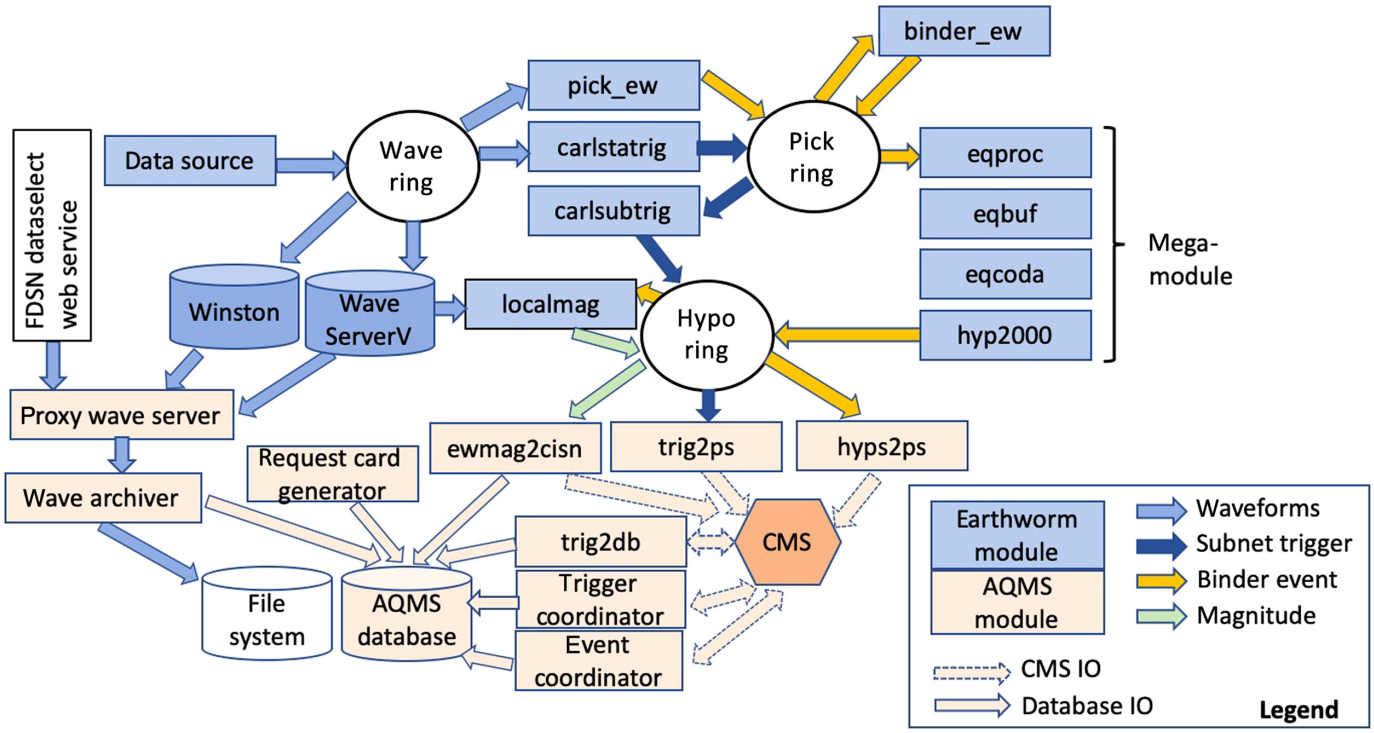

Figure 2. Diagram of message flow in a minimal AQMS setup. Modules in dark boxes indicate standard Earthworm modules, and light boxes denote AQMS modules. Two AQMS-specific earthworm modules, *trig2ps* and *hyps2ps*, send a message to the California Integrated Seismic Network (CISN) Messaging Service (CMS) message broker to notify downstream AQMS modules to take Earthworm messages and populate AQMS database tables, *trig2db* and *event coordinator*. AQMS-specific earthworm module *ewmag2cisn* writes local magnitude information from *localmag* to the database directly. The color version of this figure is available only in the electronic edition.

messages, labeled WAVE RING; a ring for exchanging pick messages, labeled PICK RING; and a ring to deposit event solutions into, labeled HYPO RING.

Two AQMS-specific earthworm modules take event messages from the HYPO RING and notify the rest of the AQMS that events were detected by publishing to the CISN Messaging Service (CMS). The CMS is an open-source Common Object Request Broker Architecture event-channel-based message broker, developed by Instrumental Software Technologies, Inc. with funding from the USGS that enables some message persistence (see Data and Resources for a link to its documentation). If one of the downstream processing modules is offline, the messages sent to CMS will not get lost and will get picked up when the module is restarted.

Module *trig2ps* takes Earthworm messages from *carlsubtrig* and writes the contents to a CMS XML message for *trig2db* to receive. It is a true Earthworm module that can be controlled and monitored by Earthworm's system processes, even though it is coded in C++. Similarly, *hyps2ps* takes Hypoinverse archive messages from an Earthworm ring and notifies the CMS subscribers. Because of limitations in the CMS system, *hyps2ps* does not actually put the Hypoinverse message into CMS. Instead, *hyps2ps* writes the message to a file in a configured directory. Then *hyps2ps* publishes a CMS message that contains the path to that file. Event Coordinator (*ec*) is the only program that may subscribe to the CMS messages from *hyps2ps*.

ec parses Hypoinverse messages from *hyps2ps* and *trig2db*, populates the database parametric information tables with this event information, and publishes CMS to notify subscribing modules about the new event. The *ec* can be compiled to parse the duration magnitude information from Hypoinverse messages and to populate that information into the parametric information tables if an Earthworm system is configured to provide duration magnitudes along with hypocenter information in the Hypoinverse message.

The program *trig2db* parses the CMS message that contains the *carlsubtrig* data, that is, a list of stations and channels involved in the subnets that triggered along with a start time of the earliest trigger and a duration to capture, and writes the information into the database. Once the data are written to the database, a CMS signal is published to be picked up by the Trigger Coordinator (*tc*) module. The *tc* module tries to reconcile coincident subnet triggers and binder events to prevent creating two event IDs for a single earthquake (duplicate). The CMS messages from *ec* or *tc* can be used by a C++ waveform request card generator (RCG) program that will write waveform requests for an event-based trigger (*Hypoinverse*) or subnet trigger (*carlsubtrig*). However, most ANSS RSNs

use a Java-based RCG that is triggered by messages stored in the database.

Local magnitudes (M_L, Richter, 1958) obtained by the Earthworm module *localmag* can be written to the AQMS database by *ewmag2cisn*. As opposed to *trig2db* and *ec*, which subscribe to CMS, *ewmag2cisn* connects directly to an Earthworm ring to subscribe to messages from *localmag*. It writes observations, magnitudes, and associated data to the AQMS database. It also calls a stored procedure to determine whether the magnitude has to be set as the preferred magnitude. Finally, it sends an event message into CMS with the ID of the event that had the M_L added to it.

ShakeMap amplitudes, energy magnitude (M_e), local magnitude (M_L)

Rapid amplitude data. AQMS provides a module called *rad2* for *rapid amplitude data* that applies several discrete time domain recursive filters (Kanamori *et al.*, 1999) to convert raw waveform data from broadband or strong-motion instruments to acceleration, velocity, displacement, response spectral (0.3, 1.0, and 3.0 s), and Wood–Anderson displacement time series. It then determines the peak amplitude within a moving time window of configurable length for each time series. The full sample rates currently allowed are 20, 40, 80, 100, 200, 250, or 500 samples per second, and the output sample rate is configurable (often set to 0.2 Hz, i.e., *rad2* measures the peak amplitude per 5 s of data). *rad2* reports peak ground acceleration (PGA), peak ground velocity (PGV), peak ground displacement (PGD), response spectral values for 0.3 s (SP03), 1.0 s (SP10), and 3.0 s (SP30) periods, peak Wood–Anderson amplitudes, ML100 (local magnitude normalized at 100 km distance, see Richter, 1958), and ME100 (energy magnitude normalized at 100 km, Choy and Boatwright, 1995; Bormann and Di Giacomo, 2011). In addition, a number of flags about the quality of the data and signal-to-noise are provided for determining if the data used were complete and not clipped. The idea behind *rad2* is to continuously compute real-time amplitudes so that other modules can collect them rapidly when an event is detected rather than having to process a large volume of waveform data all at once.

rad2 reads its raw waveform data from the Waveform Data Area (WDA) that is written to by another AQMS-specific Earthworm module *ew2wda* and writes the computed amplitudes to the Amplitude Data Area (ADA) for downstream modules *ampgen* and *trimag* to retrieve and process (Fig. 3). The WDA and ADA are shared memory segments of a fixed size that are configured to hold data for a configurable length time window, for example, 10 min long.

ShakeMap amplitudes. AQMS module *ampgen* may be activated by a CMS signal from either *ec* or *trimag* (described later). It reads the preferred origin information from the database and determines the time window during which peaks are likely to be found, then scans the ADA for peak values in that window. *ampgen* writes the PGA, PGV, PGD, SP30, SP01, and SP03 peaks for each channel to the database as well as the information that associates it to the event origin. It sends a CMS message when done. This message can be used to trigger automatic actions (see the Automatic Event Notifications and Postevent Actions section) including the start of a ShakeMap (Worden and Wald, 2016), which will retrieve the amplitudes from the database and create its usual products.

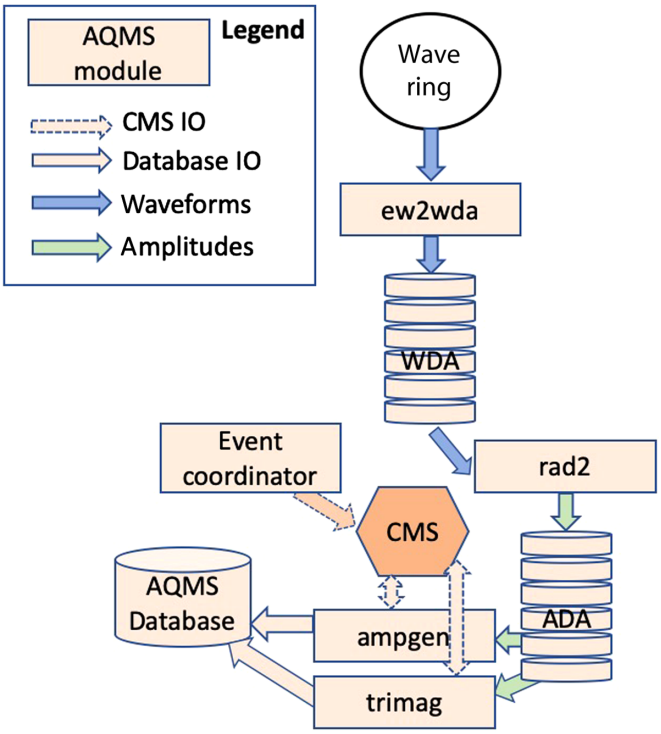

Figure 3. Diagram illustrating AQMS real-time processes for measuring peak ground acceleration, velocity, displacement, pseudospectral acceleration at periods of 0.3, 1.0, and 3.0 s, as well as ML100 and ME100 and associating them with events. See the ShakeMap Amplitudes, Energy Magnitude (M_e), Local Magnitude (M_L) section. The color version of this figure is available only in the electronic edition.

Energy (M_e) and local (M_L) magnitude. AQMS module *trimag* listens to the CMS for a new event message from the *ec*. When activated, it reads the hypocenter information from the database. The program determines a time window for each configured channel, starting at the origin time and lasting for the propagation delay (configurable) plus a data latency time (also configurable). The time window should be sufficient to bracket any ML100 or ME100 peaks for the network regardless of the event's distance. When the time window has elapsed, the program retrieves the ML100 and ME100 amplitudes from the ADA.

It is possible for *trimag* to be unable to find an M_L or an M_e (or both) for any particular event. Each magnitude calculation

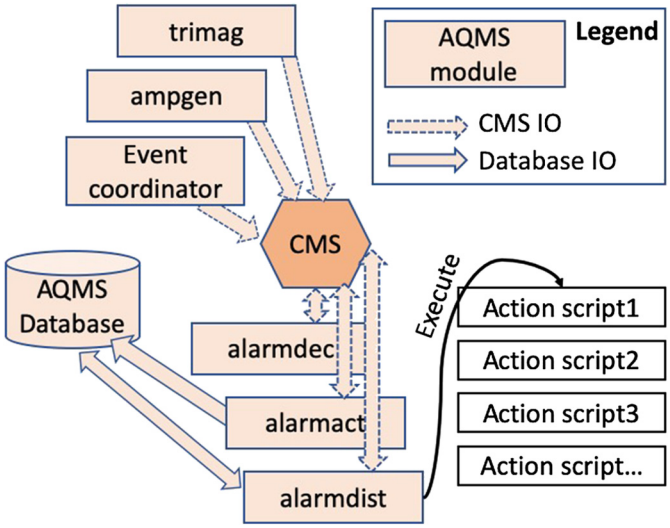

Figure 4. Diagram illustrating AQMS modules involved in sending automatic notifications. See text. The color version of this figure is available only in the electronic edition.

searches different sections of the amplitude window to find the peak, and the data may be so incomplete that the program lacks the information it needs to compute a valid magnitude. The final event M_L is a median computation over station magnitudes, and the M_e is an average of the station magnitudes. At the end of processing, the M_L and/or M_e magnitudes are written to the database and associated with the event. The on-scale amplitudes used in each magnitude calculation are saved and associated with their respective magnitudes as well. When *trimag* finishes processing an event, it sends a signal via CMS to indicate that a magnitude has been calculated. Downstream programs can pick up the signal and act upon it for alarming.

Automatic event notifications and postevent actions

A sequence of three modules manages automatic alarming and notification: *alarmdec*, *alarmact*, and *alarmdist* (Fig. 4). Notifications can be triggered by CMS signals from *ec*, *ampgen*, and/or *trimag*. The *Alarm Decision* (*alarmdec*) module subscribes to these messages. One might want one set of alarms to happen after a local magnitude is computed and a different set of alarms after *ampgen* have completed; however, one must configure a separate *alarmdec* instance per type of triggering message. In *alarmdec*, an *alarm* consists of an alarm name, together with zero or more event solution criteria. Each of the criteria must be met in order for the alarm to be declared. An event is not required to have a magnitude at the time that *alarmdec* is called to evaluate the event. However, if no magnitude is found for the event in the database, *alarmdec* will treat the event as if it had a magnitude of type "n" (no magnitude) and of value 0. A given instance of *alarmdec* may be configured for multiple alarms. Each of these alarms is evaluated independently. The success or failure of an event to meet one alarm has no effect on the other alarms. *alarmdec* can be configured to stifle some alarms during the passage of the *P* wave from a teleseism by subscribing to CMS messages from the *telestifle* program. These messages contain the start and end times of the estimated *P*-wavefront passage. Automatic event origins with times inside this window will be evaluated using an alternate set of alarm criteria. Events that are evaluated against the alternate rule set will have a log entry announcing that fact. When an alarm is declared for an event, a CMS message containing the event ID and the alarm name is published. Note that *alarmdec* does not write anything to the database.

Alarm Action (*alarmact*) decides which actions to take for a given alarm raised by *alarmdec*. Upon receiving a CMS message, it evaluates its list of configured alarm names and associated actions. For each action to be initiated, the program writes the action into the database *alarm_action* table and then notifies the alarm distribution module (*alarmdist*) via a CMS message that contains the event ID.

Alarm Distribution (*alarmdist*) is the last program in the alarming chain to run; it is also the most complicated in terms of its logic. Upon receiving a CMS message containing the event ID, it queries the database for any alarm_action entries that are PENDING or CANCELED and decides if it should operate on them with an action script or an undo script. The *alarmdist* program has two modes, DataCenter or RealTime. In DataCenter mode, it always executes the appropriate alarm scripts. In RealTime mode the *alarmdist* program checks the *rt_role* table before firing any alarm script to verify that the host on which it is running is *primary* and, if so, immediately invokes the alarm. If the host is not *primary*, *alarmdist* checks its local *peer_system_status* table to determine how to proceed. If the peer status is listed as DOWN, then the *alarmdist* will act on the action and execute the appropriate script. If the peer status is listed as OPERATIONAL, then it marks the action as OVERRULED in the database and does not invoke any alarm scripts. In practice, the *peer_system_status* is always OPERATIONAL, and there is currently no mechanism that automatically changes the *peer_system_status* table.

The standard version of *alarmdist* runs scripts sequentially in alphabetical order for all of the actions necessary for a given event. A new version, in use at two networks, allows execution of several alarm action scripts simultaneously. The scripts have a configurable, fixed number of seconds in which to perform their action, or they are deemed failed. When the *alarmdist* module starts up, it does a consistency check to see if there were any actions pending that need to be executed before it pauses and waits for new CMS messages.

Waveform archiving

Event-triggered or continuous waveforms can be archived to disk using a program called WaveArchiver (wa). wa obtains

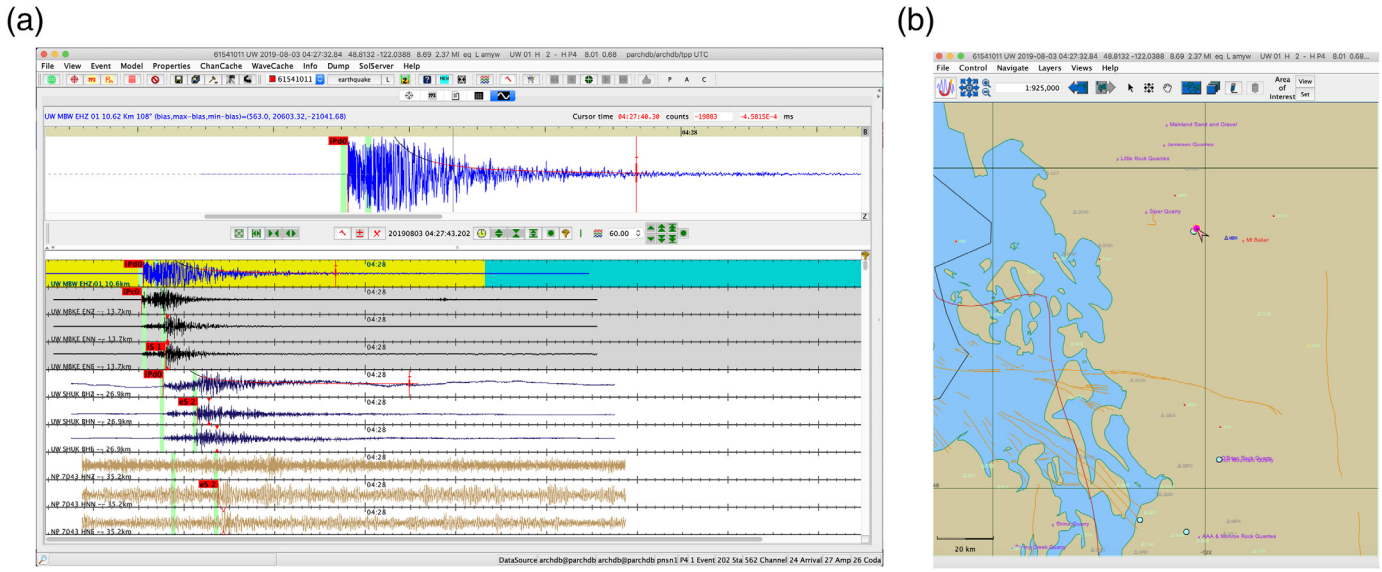

Figure 5. Screenshot of the (a) waveform and (b) map panels of the Jiggle graphical user interface as configured at the PNSN. The color version of this figure is available only in the electronic edition.

the list of channels and time windows it needs to write to disk from a table in the database called *request_card*. The *request_card* table's rows are written to the database by the RCG. Most RSNs use a Java-based RCG that gets triggered by a specific entry into the database that is created when a new event or subnet trigger is inserted into the event table. Depending on the archiving model chosen by the RCG, the set of channels and time windows is created for a given event location and magnitude. The wa requires a waveform server that provides an AQMS-specific protocol and serves data in miniSEED format. A robust, multiprocess, intermediary Proxy Wave Server (*pws*) translates between the AQMS-specific protocol and several other protocols, including the Earthworm protocol and/or the FDSN Dataselect Web Service protocol allowing the use of these more commonly used wave servers. Once the wa writes the requested miniSEED files to disk, it writes information about the waveform data, including where they are located, to the database. If successful, wa deletes the waveform request from the *request_card* table.

The waveforms can be pulled for specific events or time windows using database queries, and this is one way how Jiggle obtains its seismograms for further analysis. A limitation of this model is that the waveforms must be located on a storage system that is locally accessible to the database. Jiggle can also connect directly to the *pws* to retrieve waveforms.

Human Review and Data Extraction

AQMS provides a Java GUI-client application named Jiggle to review picks, amplitudes, and codas to be able to refine locations and magnitudes. Jiggle is highly configurable and has many features, which makes its learning curve steep. Jiggle obtains all of its earthquake parameters and waveforms from the database through Java Database Connectivity API calls over the network. To locate events, it connects to a solution service (SolServer) on the postprocessing server to run Hypoinverse. Because all interactions with Jiggle are over the network and because it is written in Java, analysts can use it on any operating system and work remotely, which is a powerful feature for most U.S. RSNs that do not have 24 × 7 operation centers. Figure 5 shows a screenshot of Jiggle as configured at the PNSN.

Quick and dirty review is done via two PHP-based web pages, the Duty Review Page (for reviewing binder events, see Fig. 6) and the Trigger Review Page (for reviewing subnet triggers).

In addition to the Jiggle GUI and the web pages, AQMS supports various mechanisms for end users to access the database. The most commonly used for catalog and phase information extraction is *dbselect*, which allows querying based on time, magnitude, location, and many other parameters related to event-based catalog information. It provides output in a variety of catalog formats in common use today. *dbselect* is a C language program. Originally, it used a proprietary Oracle-embedded SQL subsystem known as ProC to perform the database interaction between AQMS and the Oracle database. For the PostgreSQL version, *dbselect* uses ECPG, the PostgreSQL equivalent C interface. In addition, AQMS provides standard QuakeML (Schorlemmer *et al.*, 2011; see Data and Resources) output through the use of a *qml* perl program. QuakeML is sent to the ANSS Comprehensive Catalog via the USGS Product Distribution Layer (see Data and Resources). The Southern and Northern California Earthquake Data Centers have implemented webservices for AQMS to allow end users to obtain event parameters or waveforms using a standard FDSN REST interface.

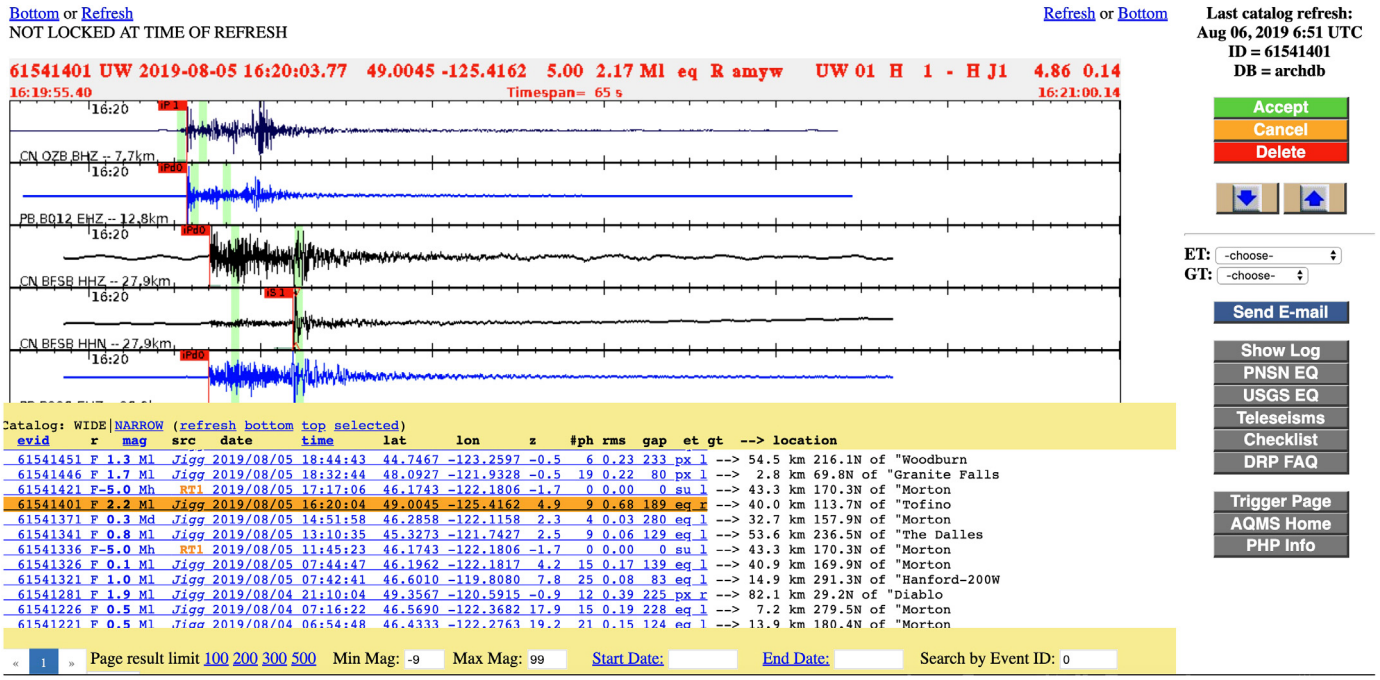

Figure 6. Screenshot of the Duty Review Page webpage showing a seismic record with labeled arrivals as a static image (upper) along with a catalog table (lower) showing the selected event highlighted. The color version of this figure is available only in the electronic edition.

Redundancy and Replication

In principle, AQMS can be run on a single server with a single database; however, when AQMS was designed, it was decided to use several AQMS servers for redundancy. A common model is to have two redundant, independent, real-time processing servers, and to isolate the databases used by the real-time processes from unpredictable loads due to human interactions and expensive queries of the complete archival catalog.

We label one real-time system as *primary* and the other as *shadow* and make sure that event IDs and other primary keys between the two systems do not clash and are uniquely identifiable as being from one server or another. In AQMS we use integer database sequences that increment by 5 to achieve unique row IDs for tables in different servers. In this scheme, only five separate database instances are possible (producing IDs ending in 0 or 5, 1 or 6, 2 or 7, 3 or 8, 4 or 9). Using Jiggle, an analyst can create a new event in the database when it was missed by the automatic systems. For this reason, the sequences on the postprocessing databases also have to follow this scheme.

Two tables in the database schema, *rt_role* and *peer_system_status*, are used to indicate whether a server is in a *primary* role or *shadow* role. Events written to a shadow database will be labeled as "not valid" by setting a flag to 0. If something was wrong with the primary real-time system during some period of time, one can promote the solutions from the shadow server during that time period to the "valid" ones simply by changing this flag to 1. In addition, for every role switch, a row is added to the *rt_role* table to indicate the new status and the time. As such, it can be tracked which machine was primary during a particular time. Several modules, for example, *alarmdist* described earlier, will check whether they are running on or are connected to a primary system to decide what action to take. Role switching of the real-time servers is a simple, though manual, process.

Transfer of data from one database to another database is achieved through database replication, which means that the data in multiple databases remain synchronized within a short time frame. Figure 7 illustrates two AQMS replication models. Figure 7a shows a diagram of the model currently in use at the PNSN, where the two real-time databases replicate event parametric data to a single archival database, which in turn replicates all its data to a hot, read-only, standby. Figure 7b shows the model previously in use at the PNSN. In this multi-master replication scheme, each real-time server replicates event parameters to a different archival database, and the two master archival databases are kept in synch through asynchronous multi-master replication.

PNSN switched replication models because stock PostgreSQL does not yet include a multi-master replication option. PNSN does not need the ability to write to two master databases at the same time, or for the postprocessing servers to work independently during a network outage. Thus, the simpler model of database interaction shown on the left works well. Enterprise editions of Oracle are capable of doing

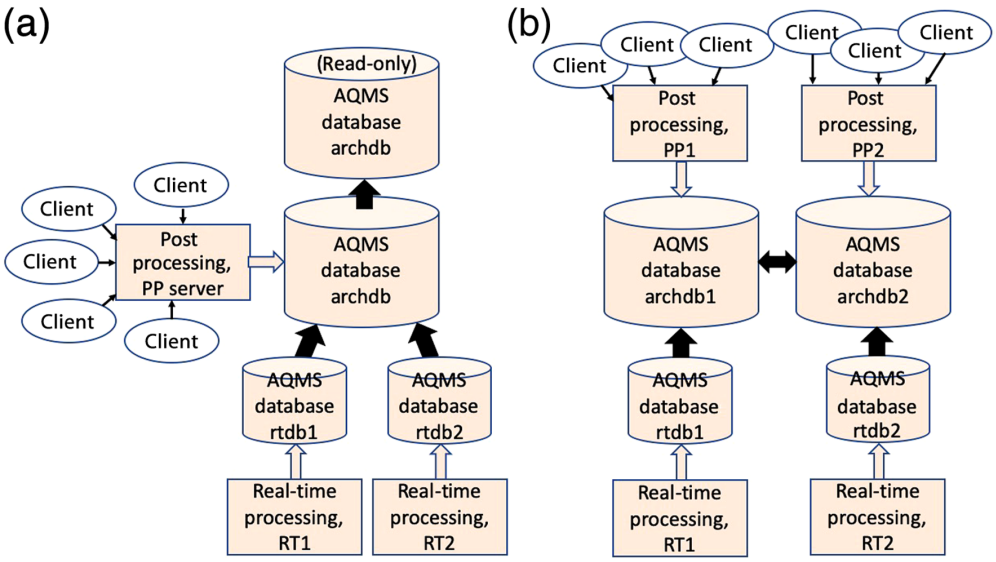

Figure 7. (a) Diagram of a single postprocessing master replication scheme that includes two redundant real-time processing systems. Replication of data from the event parameter tables in rtdb1 and rtdb2 to archdb. Replication of the complete archival database, archdb, to a hot standby. The hot standby in PostgreSQL has to be read-only. (b) Diagram of a multi-master postprocessing scheme where two archival databases (archdb1 and archdb2) can be written to, yet remain synchronized. The color version of this figure is available only in the electronic edition.

multimaster replication; however, the cost of these advanced features is significant. PNSN is using logical replication of data from the real-time databases to the master archival database and streaming replication from the archival database server to a standby database server. They implemented this system on four separate physical servers, but this could be organized differently.

In either replication model, the human analysts work with the archival databases on postprocessing servers and thus do not add load to the automated processing systems. They never need to touch the real-time databases unless there is a catastrophic failure of all postprocessing servers, in which case the real-time servers could be used as a backup postprocessing server until the databases lost in the catastrophe were recovered.

Discussion and Conclusion

By porting AQMS to PostgreSQL, the PNSN has created a free version of AQMS. Oracle database servers are time tested and known to work well at scale. The ability to use multimaster replication is useful for those networks that need multiple writable archival databases. However, Oracle database servers have so many obtusely documented features that they are difficult to administer. In addition, a significant portion of Oracle documentation is behind a paywall and accessible only if one has an Oracle support agreement. PostgreSQL is increasing its market share and is fast becoming richer in features (see Data and Resources), multimaster replication is expected to be added in a future release. The authors of this article much prefer the documentation provided by the PostgreSQL project over the Oracle provided documentation. It is beyond the scope of this article to provide a thorough comparison between Oracle and PostgreSQL; however, in sum, PostgreSQL has proved very reliable and powerful, has great documentation, and is free to use.

A relational database is more complicated to interact with than a file archive saved on a computer disk. Adding new ways to query the database requires knowledge of SQL and database programming. However, database servers can be accessed remotely, by multiple clients simultaneously. For large archives, creating complicated and detailed reports from data stores in a relational database is more efficient than from files on disk. Obtaining redundancy through database replication is powerful; however, it also requires significant time investment from RSN staff to become familiar with the setup and learn how to properly administer, maintain, and monitor the system.

AQMS is complicated to configure, but very stable once configured. In this article, we have not described a myriad of details of configuring an AQMS, but we have tried to give an overview of the overall system with enough detail to inform other RSNs of what it would take to adopt the system. One important functional part of AQMS, the orchestrated interaction between the database and external programs on the postprocessing side has been alluded to, but we refer to the AQMS documentation for further details.

PostgreSQL AQMS has been in production for close to a year at a single RSN and as such, not all AQMS features have been thoroughly tested, nor have all add-on modules been ported. Recently, PostgreSQL AQMS was rolled out to the Puerto Rico Seismic Network, and we hope that, as more people use the system, it will get thoroughly tested and vetted. AQMS source code and documentation are available at the gitlab website (see Data and Resources), where bug reports can be filed as well.

Data and Resources

No data were used in this article; however, many resources were referred to and are listed as follows: Advanced National Seismic

System (ANSS) Quake Monitoring System (AQMS) source code: https://vault.gps.caltech.edu/trac/cisn/wiki (last accessed September 2019, to become obsolete), or www.gitlab.com/aqms.swg (last accessed September 2019) or http://aqms.swg.gitlab.io/aqms-docs (last accessed September 2019). AQMS database schema: www.ncedc.org/db or www.gitlab.com/aqms.swg/aqms-db-pg/create (last accessed October 2019); Earthworm: http://www.earthwormcentral.org/ and http://www.earthwormcentral.org/documentation4/ovr/carltrig_ovr.html (last accessed September 2019); Hypoinverse: ftp://ehzftp.wr.usgs.gov/klein/hyp1.41/ (includes documentation) or ftp://ehzftp.wr.usgs.gov/klein/hyp1.42 (latest source code, last accessed September 2019); California Integrated Seismic Network (CISN) Messaging Service (QWServer): http://www.isti.com/QWIDS/current_dist/QWServer/doc/QWServer.html#notif (last accessed September 2019); getStationXML: https://github.com/pnsn/aqms-ir (last accessed October 2019); FDSN StationXML specification: https://www.fdsn.org/xml/station/ (last accessed September 2019); QuakeML specification: https://quake.ethz.ch/quakeml/ (last accessed September 2019); SEISAN: http://seis.geus.net/software/seisan/seisan.html (last accessed September 2019); SeisComp3: https://www.seiscomp3.org/ (last accessed September 2019); ANSS Comprehensive Catalog: https://earthquake.usgs.gov/data/comcat/contributor/ (last accessed September 2019); U.S. Geological Survey's Product Distribution Layer: https://usgs.github.io/pdl/ (last accessed September 2019); ANSS Station Information System: https://wiki.anss-sis.scsn.org/SIStrac (last accessed September 2019); Oracle vs. PostgreSQL comparison: https://db-engines.com/en/system/Oracle%3bPostgreSQL (last accessed September 2019). PostgreSQL feature matrix website can be accessed at https://www.postgresql.org/about/featurematrix/ (last accessed September 2019). The Geoserve (Fig. 1) can be accessed at https://earthquake.usgs.gov/ws/geoserve/layers.php (last accessed September 2019).

Acknowledgments

The authors are grateful to all the programmers who have worked on Earthworm and Advanced National Seismic System (ANSS) Quake Monitoring System (AQMS), but in particular: Aparna Bhaskaran, Shang-Lin Chen, Andrew Good, Peter Lombard, Phil Maechling, Doug Neuhauser, Patrick Small, Kalpesh Solanki, Ellen Yu, and Stephane Zuzlewski. The authors thank two reviewers for their prompt reviews and thorough reading of the original article. Funding for the development of AQMS has come from the U.S. Geological Survey, the state of California, California Institute of Technology, University of California Berkeley, and the University of Washington. The Gordon and Betty Moore Foundation funded the PostgreSQL transition.

References

Bormann, P., and G. Di Giacomo (2011). The moment magnitude Mw and the energy magnitude M_e: Common roots and differences, *J Seismol.* **15,** 411–427, doi: 10.1007/s10950-010-9219-2.

Choy, G. L., and J. Boatwright (1995). Global patterns of radiated seismic energy and apparent stress, *J. Geophys. Res.* **100,** 18,205–18,228, doi: 10.1029/95JB01969.

Friberg, P., S. Lisowski, I. Dricker, and S. Hellman (2010). Earthworm in the 21st century, *Geophys. Res. Abstr.* **12,** 12,654.

Hanka, W., J. Saul, B. Weber, J. Becker, P. Harjadi, Fauzi, and GITEWS Seismology Group (2010). Real-time earthquake monitoring for tsunami warning in the Indian Ocean and beyond, *Nat. Hazards Earth Syst. Sci.* **10,** 2611–2622.

Havskov, J., and L. Ottemöller (1999). SeisAn earthquake analysis software, *Seismol. Res. Lett.* **70,** no. 5, 532–534. doi: 10.1785/gssrl.70.5.532.

Johnson, C., A. Lindh, and B. Hirshorn (1997). Robust regional phase association, *U.S. Geol. Surv. Open-File Rept. 94–621.*

Johnson, C. E., A. Bittenbinder, B. Bogaert, L. Dietz, and W. Kohler (1995). Earthworm: A flexible approach to seismic network processing, *IRIS Newsl.* **14,** no. 2, 1–4.

Kanamori, H, P Maechling, and E. Hauksson (1999). Continuous monitoring of ground motion parameters, *Bull. Seismol. Soc. Am.* **89,** no. 1, p311–316.

Klein, F. W. (2002). User's guide to Hypoinverse-2000, a FORTRAN program to solve for earthquake locations and magnitude, *U.S. Geol. Surv. Open-File Rept. 02–171*, 121.

Lomax, A., A. Michelini, and A. Curtis (2009). Earthquake location, direct, global-search methods, in Complexity, in *Encyclopedia of Complexity and System Science, Part 5*, R. A. Meyers (Editor), Springer, New York, 2449–2473, doi: 10.1007/978-0-387-30440-3.

Lomax, A., J. Virieux, P. Volant, and C. Berge (2000). Probabilistic earthquake location in 3D and layered models: Introduction of a Metropolis–Gibbs method and comparison with linear locations, in *Advances in Seismic Event Location*, C. H. Thurber and N. Rabinowitz (Editors), Kluwer, Amsterdam, The Netherlands, 101–134.

Olivieri, M., and J. Clinton (2012). An almost fair comparison between Earthworm and SeisComp3, *Seismol. Res. Lett.* **83,** 720–727, doi: 10.1785/0220110111.

Quintiliani, M., and S. Pintore (2013). Mole: An open near real-time database centric Earthworm subsystem, *Seismol. Res. Lett.* **84,** no. 4, 695–701.

Richter, C. F. (1958). *Elementary Seismology*, W. H. Freeman, San Francisco, California.

Schorlemmer, D., F. Euchner, P. Kästli, and J. Saul (2011). QuakeML: Status of the XML-based seismological data exchange format, *Ann. Geophys.* **54,** no. 1, 59–65.

Worden, C.B., and D. J. Wald (2016). ShakeMap manual online: Technical manual, user's guide, and software guide, *U. S. Geol. Surv.*, doi: 10.5066/F7D21VPQ.

Yu, E., P. Acharya, J. Jaramillo, S. Kientz, V. Thomas, and E. Hauksson (2017). The Station Information System (SIS): A centralized repository for populating, managing, and distributing metadata of the advanced national seismic system stations, *Seismol. Res. Lett.* **89,** no. 1, 47–55, doi: 10.1785/0220170130.

Manuscript received 9 August 2019
Published online 20 November 2019