Fast Playback Framework for Analysis of Ground-Based Doppler Radar Observations Using MapReduce Technology

JINGYIN TANG AND CORENE J. MATYAS
Department of Geography, University of Florida, Gainesville, Florida

(Manuscript received 23 May 2015, in final form 12 January 2016)

ABSTRACT

The creation of a 3D mosaic is often the first step when using the high-spatial- and temporal-resolution data produced by ground-based radars. Efficient yet accurate methods are needed to mosaic data from dozens of radars to better understand the precipitation processes in synoptic-scale systems such as tropical cyclones. Research-grade radar mosaic methods of analyzing historical weather events should utilize data from both sides of a moving temporal window and process them in a flexible data architecture that is not available in most stand-alone software tools or real-time systems. Thus, these historical analyses require a different strategy for optimizing flexibility and scalability by removing time constraints from the design. This paper presents a MapReduce-based playback framework using Apache Spark’s computational engine to interpolate large volumes of radar reflectivity and velocity data onto 3D grids. Designed as being friendly to use on a high-performance computing cluster, these methods may also be executed on a low-end configured machine. A protocol is designed to enable interoperability with GIS and spatial analysis functions in this framework. Open-source software is utilized to enhance radar usability in the nonspecialist community. Case studies during a tropical cyclone landfall shows this framework’s capability of efficiently creating a large-scale high-resolution 3D radar mosaic with the integration of GIS functions for spatial analysis.

1. Introduction

Overview of 3D weather radar mosaic

Ground-based weather radars sample the atmosphere at a high spatial and temporal resolution, making them the preferred instrument for remotely sensing precipitation over land (Germann and Zawadzki 2002). In the United States, the Weather Surveillance Radar-1988 Doppler (WSR-88D), or Next Generation Weather Radar (NEXRAD), a network of S-band radars, has been operational since 1995. Approximately 160 radars provide more than 20 years of data for atmospheric research (Crum and Alberty 1993). However, the mosaicking of data from dozens of radars to analyze synoptic-scale systems, such as tropical cyclones (TCs), presents challenges in computational efficiency and accuracy, especially for researchers who are not formally trained in radar meteorology. For example, Matyas (2007, 2010) did not consider altitude when utilizing a geographical information system (GIS) to mosaic base reflectivity data, calculate shape metrics, and measure the sizes of rainfall regions in landfalling TCs.

Because of the scanning limitations of a single radar (Carbone et al. 1985), multiradar composites are essential to many meteorological applications, including real-time weather forecasting, observational research of synoptic-scale weather systems, and numerical weather prediction models. This range of applications requires different data processing strategies. Real-time analysis requires a time-sensitive handling of input data from the recent past. On the other hand, research-grade applications can utilize data on both sides of a moving temporal window to construct the best possible representation of precipitation structures in the atmosphere. For this purpose, multiple pieces of software and libraries were developed (Ansari et al. 2010). Based on the software designing mode, they could be categorized into three types: 1) single-node tools, 2) service clients, and 3) batch processing systems. Single-node tools such as Radx (Dixon and Wiener 1993), SOLO (Nettleton et al. 1993), and Reorder (Oye and Case 1995) enable researchers to run software locally using personal computers or...
workstations to decode, extract, and transform archived radar data. Visualization of radar data on screen is also supported by software, such as the Weather and Climate Toolkit (Ansari et al. 2010). Service clients like the Virtual University of Chicago–Illinois State Water Survey (VCHILL; Chandrasekar et al. 2005) preprocess and prepare radar data using data services provided by a remote data server. These clients require a low amount of computing resources on researchers’ local computers. Batch processing systems are usually designed for processing large volumes of data in parallelized architecture, optionally distributed over multiple nodes. Real-time processing systems and warning decision systems (Michalakes et al. 2004; Gao et al. 2013; Lakshmanan and Humphrey 2014) fall into this category. They must run on a specialized platform to ensure data are processed at real-time speeds. For example, platform guides for the Warning Decision Support System—Integrated Information proposes dedicated high-performance disks and 16-GB memory per CPU core (Lakshmanan et al. 2007b). Although these three categories handle radar data at different scales, these requirements are not optimal for research-grade applications or feasible for use by the community outside of meteorology, thus limiting the knowledge that could be gained through analysis of these high-resolution data.

Because of the conical nature that radar senses objects in the atmosphere, radar data are usually sampled, recorded, and stored in uncommon binary formats (e.g., NEXRAD archive for the WSR-88D archive, DX format for the German Radar Service, Common Doppler Radar Exchange Format (Universal Format) and Climate and Forecast (CF)-compliant netCDF for general radial radar data). Thus, before radar data can be analyzed for research purposes, multiple preprocessing steps are required. Since radar antennas rotate, weather radar data are recorded in polar coordinates (azimuth, elevation, and range). The data must be mapped to Cartesian coordinates to fit the needs of applications that require radius-insensitive values and constant-sized gridded data (Mohr et al. 1986). Because of the curvature of the radar beam and the earth’s surface, the range of a single radar is limited to 460 km at the lowest tilt in the case of U.S. WSR-88D radars and 230 km in higher tilts. To analyze synoptic-scale systems such as TCs, it is necessary to mosaic data from multiple individual radars onto a Cartesian grid. Because weather systems move over time, it is preferable to process and display weather radar data seamlessly over large domains, regardless of the radar from which the data originated (Vasiloff et al. 2007; Lakshmanan et al. 2007b). With the aid of high-performance computing and parallel processing, 3D radar mosaics can be constructed faster than real time to permit the analysis of multiple past weather events.

WSR-88D radar data from National Climatic Data Center are stored in two formats: level II archives and level III products (Crum et al. 1993). Level II data include basic Doppler radar moments and system status information. They contain three important variables—reflectivity, radial velocity, and spectrum width—and are encoded as a unique binary format. Additionally, differential reflectivity, correlation coefficient, and differential phases are available in dual-polarized level II data. As level II data do not always accurately and directly portray information for hydrometeors, they are mostly used by trained meteorologists. Level III products are derived from level II archives and are preprocessed to facilitate their use by nonspecialist researchers (Matyas 2010). But level III products, such as base reflectivity and composite reflectivity, have limitations regarding elevation that do not allow the vertical profile of raining areas to be discerned. A better solution is to create a product using algorithms similar to those for level III but that yield output for multiple vertical levels and that are less complicated to perform. The need for this type of product was confirmed through a survey conducted by the National Center for Atmospheric Research during an international workshop. The survey showed a high demand for a methodology of integrating radar data with nonradar data (such as satellite images, land-use data, topology) using open-source software that could support both personal research and community usage (Bluestein et al. 2014). Providing a straightforward 3D interpretation of radar data stored as an uncomplicated file format will enhance research by permitting a larger group of scientists to more easily access the data.

To fulfill these demands, the goal of this research is to develop a method to create a mosaic with data from multiple radars and to perform spatial analysis in the most computationally efficient manner possible. In this paper, we present a fast playback system for creating a 3D mosaic of reflectivity and wind velocity using level II data from multiple WSR-88D radars. This system provides a scalable, computationally efficient, geospatial, and GIS-friendly environment. We present a flexible and extensible method to incorporate radar mosaic technology into a popular open-source parallel computational framework (Apache Spark) and industrial-level GIS software (ArcGIS) to enhance efficiency and simplicity.

2. Advantages of diagnostic playback over real-time processing

We will utilize the term diagnostic playback to refer to the use of radar data to examine past weather events.
Both real-time processing and diagnostic playback systems collect observations from one or multiple radar stations, place them in the correct position on the research domain, construct a 3D mosaic, and compose radar products. Although both systems share the same methodology, there are different considerations for three elements: timeliness pressure, data availability, and system runtime environment. These differences are highlighted below.

Time is critical in a real-time system. The real-time system must process radar data as soon as they are streamed to the system, and the minimum processing speed must be as fast as real time because processed data are immediately required as inputs to follow-up procedures, especially in a warning decision scenario. Thus, a real-time system usually places a temporal margin at each stage to safely handle unexpected latencies, for example, an input/output bottleneck in hard disks, or the system might be blocked by the operating system until old data are processed and flushed out. Avoiding such latency during an operational run necessitates that the design of a real-time system focuses more on stability than flexibility. With the hard constraints of running time, algorithms for real-time systems also must balance accuracy, complexity, and available computing resources. Thus, a real-time system often needs a dedicated environment to ensure that algorithms run timely and to reduce latency in all steps, especially while processing data from multiple radars when the volume of data streaming is large. In contrast, because there is no time pressure for running a diagnostic case, a playback system can run in a different style by prioritizing flexibility and scalability. Without hard time constraints, a playback system leads to lower requirements on both hardware and software environments, and it can be run on an arbitrary system capable of completing a finite case in a user-acceptable duration. Also, complicated algorithms and experiments could be executed without consideration for time pressure. Latency in a playback system can be handled by either slowing down the running speed or hanging up the system until all queuing data are processed.

Two additional details also distinguish historical analysis from that in real time: finite experiment duration and the need to make changes. Before commencing playback of a historical case, all data are available and the exact end time for the case is known. Thus, algorithms applied in a playback system can produce a two-sided temporal interpolation at any single moment by using past and future data. Also, the researcher can adjust the range of possible times to be included in the output product in both directions. However, this is not always possible in a real-time system when processing the most leading time data because it is impossible to acquire observations in future time. So, algorithms in a real-time system are usually limited to a one-sided temporal interpolation. Furthermore, the playback process is usually carried out by researchers who require multiple experiments with parameters that may change frequently. A playback system is a more suitable platform for these tests than a real-time system. The system runtime environment also requires the flexibility to easily extend the functionality in the system to accommodate such changes. Given this, the playback system needs a different type of architecture rather than simply porting a real-time system to emulate a diagnostic case in the same manner as it would process a real-time case.

3. Playback system architecture

As the previous section highlighted the demands for research-grade operations, our playback system is designed on a distributed computation environment. We construct our system using the concept of an actor model, and implement it with Apache Spark and ArcGIS Runtime using the Scala programming language. This combination considers feasibility in its implementation in addition to scalability, flexibility, and computational efficiency.

a. Parallelization principle and software

Admittedly, batch processing large volumes of radar data could be done via naive parallelizing at high levels using existing open-source software [e.g., Radx, Weather Climate Toolkit, Python ARM Radar Toolkit (Py-ART)] by utilizing high-level application programming interface (API) or command line interface (CLI). For example, one could use GNU Parallel to parallelize several instances of Radx CLI commands, each processing a list of input files. This scheme is less optimal in three main aspects. First, most existing software is naturally designed to operate on a single workstation. Second, although they provide a fully functional interface in an easy-to-use style, these functions are mostly designed to be run inside a stand-alone application in a single-thread loop. Such a long sequential loop performs inefficiently in a distributed environment. Finally, it is nearly impossible to exchange data efficiently between multiple instances of one application, even in one local node. These aspects lead to an impossible task of coordinating the computation of large volumes of data among multiple computers (e.g., high performance cluster), which is often accessible by many researchers. Thus, the coordination of computation is essential to parallelize radar processing procedures in a low level. The principle in our proposed scheme is to divide a procedure into small units. Each unit contains a simple small calculation with minimal data dependencies. Those units are stored in a pool. A unit will be picked out from the pool and executed by any idle...
Parallelization in this framework is based on Apache Spark. Apache Spark is a fast and general engine for large-scale data processing capable of executing our commands in an efficient manner (Dean and Ghemawat 2008) where the functions of mosaicking radar and spatial analytics are plugged in and executed. Apache Spark provides high-level programming interfaces in Java, Scala, and Python programming languages. It also supports running on a single node as a simple concurrent execution engine and as a cluster across multiple nodes, allowing our system to be run by users with or without access to high-performance computing clusters.

In this research, we utilize an actor model, the fundamental communication and parallelization mechanism in Apache Spark, to simulate a radar network composed by radars involved in the playback case. The simulation is straightforward: each actor acts as an individual radar station in the network, actors change their respective status when the corresponding radar station changes its operating mode by external control or by self-controlled rules. A radar actor continuously sends out messages that contain observations to the processor actor until the case ends or it receives stop commands from supervisors. During the simulation, those individual actors preserve time asynchrony among multiple radar stations. The simulation of the radar network is not a mandatory requirement in the presented parallelization scheme, but rather it simplified the procedure to adopt an optional radar station–specific algorithm on the corresponding radar actor in the parallelization scheme.

**b. Radar data mosaic methods**

The level II data are stored in a spherical coordinate system. Thus, the first challenge in creating a 3D mosaic domain is to transform the coordinates into a Cartesian 3D grid. In a single-radar scenario, a gate with a slant range to its corresponding radar station equal to $S_r$, at an azimuth of $\alpha$ and an elevation of $\phi$, the relationship of its position in a radarcentric conic coordinate system to its position $(x, y, z)$ on a Cartesian grid is:

$$
\begin{align*}
  x &= S_r \times \cos\phi \times \cos\left(\frac{\pi}{2} - \alpha\right) + x_r, \\
  y &= S_r \times \sin\phi \times \sin\left(\frac{\pi}{2} - \alpha\right) + y_r, \\
  z &= H + z_r, \\
  H &= S_r \times \sin\phi + \frac{S_r^2}{2 \times I_r \times R_e}, \\
  S_{rg} &= r_g \times (i - 1) + r_0
\end{align*}
$$

where $(x_r, y_r, z_r)$ are the coordinates of the radar antenna in a 3D Cartesian coordinate system, $H$ is the height of the beam centerline above the radar level (km) given by WSR-88D operational beam propagation, $I_r$ is the standard refractive index equal to 1.21, the radius of earth $(R_e)$ is set as 6371 km, and $S_r$ is the slant range of the $i$th gate in a beam, where $r_g$ is the gate width (1 km in legacy radar products: 0.25 km in superresolution radar products) and $r_0$ is the distance to the first gate. Super-resolution radar products became available during 2008 at most WSR-88D sites (NOAA/ROC 2008).

The above-mentioned equations assume that the earth’s surface is flat. To combine data from multiple radars onto a single grid, the earth’s curved surface needs to be projected onto a flat surface. The choice of projection system is based on the extent of the research domain and spatial distribution of radar stations. To maintain the circle shape of the radar scan and to preserve the true direction of all azimuths, conformal projection methods are preferred. With properly defined parameters, the projection will not yield large spatial errors. Distance errors are less than 1.3% when using the Lambert conformal conic projection that covers the continental United States with central latitude at 39°N and standard parallels at 33° and 45°N. Errors will be less for smaller research domains.

**c. Introduction to resilient distributed dataset and MapReduce**

MapReduce is a programming model for parallelizing data processing across massive numbers of computers in a distributed environment (Dean and Ghemawat 2008). MapReduce achieves large-scale data processing for parallel computing by using map and reduce functions, prohibiting the data exchange between map functions in the map stage and reduce functions in the reduce phase. As shown in Fig. 1, each MapReduce job takes inputs as a key-value dataset, producing a new key-value dataset with the following steps:

- The map phase executes map functions, which at each time accepts one key-value pair and transforms it into one or multiple intermediate key-value pairs: $\{ (k_1, v_1) \} \rightarrow \{ (k_2, v_2) \}$.
- The shuffle phase collects all intermediate key-values pairs from the map functions, sorts them by keys, and groups values with the same keys: $\{ (k_2, v_2) \} \rightarrow \{ k_2, [v_2] \}$.
- The reduce phase aggregates grouped values and produces one key-value pair for each group: $\{ k_2, [v_2] \} \rightarrow \{ k_3, v_3 \}$.

All inputs, intermediate results, and outputs are always represented as key-value pairs. The key-value pairs are stored in resilient distributed datasets (RDDs)—fundamental elements that carry and transform data.
in Apache Spark. RDDs should be kept small in order to be stored and exchanged completely in memory for high performance, as suggested in Apache Spark programming guides (Apache 2015). For our routine, each RDD contains a small portion of scans at one elevation for one radar. Initially, each RDD is tagged by the time of echoes and the coordinates in the research domain, which are then converted into a spatial–temporal index as keys. Once the spatial–temporal keys are defined, they fit the requirement of the MapReduce paradigm. Apache Spark is advantageous as it is possible to chain multiple map and reduce functions in a pipeline, which enables us to operate on complex tasks in MapReduce jobs. Because exchanging data between nodes consumes resources and time, the restriction of data exchange inside each stage guarantees high performance when scaling up the system.

d. Processing radar data using RDDs

A system run begins from reading environmental configurations. The configurations initialize variables of the following: definition of geographic projection; positions of known radar stations; start and end times of a diagnostic case; spatial and temporal resolution of mosaicking datasets, which determines spatial–temporal keys; name of spatial analysis functions to be executed; and total available CPUs and memory available. These environmental configurations must be deployed to all nodes in a cluster before MapReduce functions are executed on separate nodes because no data are exchanged between map functions or reduce functions on different nodes. After the environmental initialization, the system is ready to commence MapReduce functions. The complete procedure of playing back a diagnostic case includes four steps: preprocess, map function chain, reducing function chain, and postprocess (Fig. 2). These steps are detailed below for our radar research task.

1) PREPROCESS

In this step raw level II archives are decompressed to dump into the file system. Then they are decoded and split into small parts according to the names of variables [i.e., reflectivity (dBZ), radial velocity, and spectral width], scan elevation, and time, and then written to individual netCDF (Network Common Data Form) files (Rew and Davis 1990). An optional quality control algorithm for radar data could be employed here. For example, utilizing the w2gcncdnp (Lakshmanan et al. 2007a) application from WDSS-II, a real-time radar processing system (Lakshmanan et al. 2014) for level II radar data could be beneficial (e.g., removing clutter, sun spikes). Velocity–azimuthal display (VAD) technologies are employed here to add horizontal wind information. Although multiple improvements are proposed based on VAD in later research (Gao et al. 2004a; Matejka and Srivastava 1991), VAD employed at this stage is for providing only an initial guess that will be corrected in a later stage. The netCDF files are further dumped and converted to JavaScript Object Notation (JSON) as it is a compact text and is compatible with Apache Spark (Crockford 2006). The converted JSON files store attributes and metadata as JSON dictionaries and variables as JSON lists. These JSON files are the initial inputs to the map function chain. For dual-polarization radar data, specific differential phases ($K_{dp}$) are also calculated based on the differential phase shift ($\phi_{dp}$) using the convolution filter presented in Vulpiani et al. (2012).
It is also beneficial to adopt a MapReduce step at preprocessing stage to parallelize VAD and $K_{dp}$ calculation. For the VAD algorithm, the minimal input set is a sweep of gates at one elevation and one gate range, and for the $K_{dp}$ convolution filter, the inputs will be individual beams. So the calculations are easily mapped into multiple mapping functions, while the reducing functions are null because these calculations add additional information to the original gates.

2) MAP FUNCTION CHAIN

Our utilization of Apache Spark permits the flexibility to connect multiple functions as a whole map chain so long as the output formats after the last function are formatted as key-value pairs. In the map stage, we chain three mapping steps. The goal of the map stage is to tag all gates sampled by radar devices with their correct spatial temporal keys. The steps are as follows.

Initial inputs from the preprocessor are individual JSON files that contain a complete 360° scan at one antenna elevation. Each file is read into an RDD, and the key of the RDD is the start time of the scan at that elevation. Then individual beams are extracted from that scan and saved into individual beam-RDDs and are tagged by the exact sampling time. The time is calculated from the rotation rate of its corresponding volume coverage pattern (VCP) and scan start time. The next step is to identify which data are needed for analysis at a predetermined time. The temporal resolution of the final output may not perfectly match the radar scan period and at least one completed volume scan must be collected from each radar to generate a complete 3D mosaic. Thus, RDDs may need duplication if they are needed for multiple outputs over short time intervals or removal if the output time is long. When the start time and temporal resolution are given, the time is mapped to an integer starting from index 0. RDDs with the same time index are gathered in the reduce stage to calculate the 3D mosaic.

Then each beam-RDD is decomposed gate by gate into multiple gate-RDDs. Next, the gate’s polar coordinate $(r, \alpha, \phi)$ will be transformed to its true coordinate $(x, y, z)$, and then further mapped to a grid index of $(i, j, k)$ according to the domain’s original points and spatial resolution as defined in the configuration. Only the gate center point is mapped during the transformation. Empty grid cells exist when the domain resolution is high. Filling blank cells exist when the domain resolution is high. Filling blank cells will be done in later reducing functions. Combined with the temporal index from the previous step,
the final key \((T, i, j, k)\) contains \(T\) as the temporal index and \((i, j, k)\) as the spatial index. RDDs that contain the same key fall into the same grid point in one 3D mosaic dataset at one time.

In the map stage, one RDD is always “mapped” into multiple RDDs in each step: an input-RDD reads an input file that contains a complete sweep of the elevation scan that is then is mapped into multiple beam-RDDs; each beam-RDD is then mapped into multiple gate-RDDs. No messages are exchanged between RDDs inside each step, thus allowing input-RDDs to be randomly dispersed among multiple nodes to create beam-RDDs, and so forth. Tagging all RDDs with keys guarantees correctness in the map stage.

3) SHUFFLE AND REDUCING FUNCTION CHAIN

After the map step, Apache Spark collects all RDDs that were calculated across multiple nodes and sorts them by keys. The shuffle stage groups RDDs first by key \(T\) and then key \((i, j, k)\). Each group serves as inputs to the reduce step. Similar to the map function chain described previously, it is possible to have multiple functions in the reduce chain. Although variables of reflectivity and radial velocity are mapped at the same time, they are aggregated separately in the reduce stage because different methods are applied to construct their mosaics. Furthermore, mosaicked reflectivity datasets are required by the method employed in this paper to derive wind velocity vectors from scalar radial velocity. So, we calculate the reflectivity product first, followed by the wind velocity product.

4. Mosaicking radar data

a. Combining scalar and vector radar variables

The processing of reflectivity data is straightforward. For one temporal key, all RDDs tagged with this key are combined into one 3D mosaic dataset. It is a common occurrence for a grid cell to lack an observation when it is located in the gap between two radar beams at high elevation and close to the radar. So, we first compute grid values that contain nonzero observations and then use those values to interpolate values for grid points with no observations.

When more than one observation exists for a grid point, the grid value is computed as a weighted average because an echo is more accurate when it is detected where the beam energy density is higher and its observation time is closer to the output time of the mosaic dataset. A time–distance-weighted function proposed by Lakshmanan et al. (2006) is employed here. After this step, all points containing at least one observation are filled. When no observations are available with which to populate the grid, the point is marked as “missing” but no further action is performed at this step. We follow the same procedure to combine other scalar variables and dual-polarization variables, for example, differential reflectivity \((Z_{dr})\) and specific differential phase \((K_{dp})\).

After the preliminary mosaic dataset is created, the results are sent to a geospatial function where interpolation is used to fill in the missing values. A distance-weighed function is employed so that missing values are averaged based on the surrounding values. A \(k\)-dimensional (k-d) tree (Bentley 1975) is built to accelerate nearest-neighbor searching. The horizontal distance is weighted 2.5 times the vertical distance. The purpose in adopting different factors is to aid in overcoming the problem of the sampling gaps at high altitudes near the radar that produces a bull’s-eye pattern of concentric rings, especially when bright bands are present.

The procedure of combining velocity data is more complex. True wind velocity over a domain is a three-order array filled by a 3D vector. Since the VAD technology employed above gives only horizontal wind direction, the mosaicked 3D wind grid also contains only a horizontal wind field at each vertical level. To fully utilize both 3D reflectivity and horizontal wind fields, we further implemented a three-dimensional variation (3DVAR) data assimilation method proposed by Gao et al. (2004b) to analyze vertical winds and to make corrections on horizontal winds. The implementation is to minimize an overall cost function equal to the square sum of the difference between the calculated radial velocity to radar stations from the wind mosaic and radar-observed radial velocity, plus a penalty term of a weak anelastic mass continuity. The 3D reflectivity grid provides the required reflectivity to calculate hydrometeor terminal velocity based on the formula in Montmerle et al. (2001).

The initial guess is the wind mosaic with all vertical wind speeds set to 0. We apply a \(3 \times 3 \times 3\) moving cubic to parallelize the calculation of Jacobian and Hessian terms so that the optimized 3D wind field can be efficiently obtained using the quasi-Newton method.

b. Communicating with geospatial analytics functions

Previous research has shown that the use of GIS can enhance the understanding of radar mosaic images by measuring shape metrics and applying geospatial analytics (Matyas 2007; Hu 2014; Tiranti et al. 2014). For example, tracing shape changes of rain fields can help reveal the interaction between TCs and topographical features or middle latitude weather systems (Lin et al. 2002). Although many geospatial analytic functions are based on computational geometry and statistics and could be ported into the system using Scala and Java,
it is better to utilize existing GIS software rather than rewrite the same GIS functions. Thus, we employ ArcGIS Runtime to integrate with the system for three main reasons: 1) The geospatial analytics function—namely, “geoprocessing” in ArcGIS software—is considered to be robust (Steiniger and Bocher 2009). 2) ArcGIS Runtime contains a Lightware core that can be deployed easily on multiple nodes without a complicated configuration and heavy dependencies on graphics user interface libraries. 3) ArcGIS Runtime provides a Java-based software development kit (SDK) capable of collaborating with Scala and Apache Spark. Our method also allows users to replace the ArcGIS part with similar implementations as long as these functions are able to accept and return key-value pair RDDs.

To enable interoperation between Spark and ArcGIS Runtime, a scriptable interface is designed to pass arguments and values between RDDs in Spark and ArcGIS Runtime. The scriptable interface triggers a local geoprocessing server if no local server is running on the current node. It then submits geoprocessing tasks to the local server, waits until execution finishes and retrieves the results, and transforms them as RDDs again. The scriptable interface permits the ArcGIS Runtime to run as a stand-alone program as a workaround of potential license conflicts with the entire open-source playback system.

The officially documented interface in ArcGIS Runtime is not suitable for executing arbitrary geoprocessing tasks because a local geoprocessing server is able to execute only a predefined “geoprocessing package.” However, ArcGIS Runtime is able to execute any Python script and it is possible to call all geoprocessing functions inside the Python script. This gives the scriptable interface in the system a proper method to overcome this limitation. On the playback system’s side, the scriptable interface generates Python codes containing all geospatial analytics functions dynamically, and then submits codes and necessary data to the local server. On the local server’s side, a specially designed geoprocessing package is deployed to simply execute the Python codes it received. For our system, ESRI ASCII Grid is chosen to carry all raster datasets. Both JSON is chosen to carry all vector datasets and ESRI VELOCITY mosaics are derived directly from grouped RDDs rather than stacked from 2D slices. Thus, the 3D mosaic must be created first before taking a slice through the mosaic to create a 2D cross section even though only 2D slices are desired, as shown in many studies (Villarini et al. 2011).

5. Case demonstration

Our playback system has been applied to reconstruct 30 TC landfall cases from 1995 to 2015, spanning both legacy and superresolution (dual polarization included) level II datasets. The number of radars involved in these cases ranges from 8 to 28. Each case started from the time when a TC’s center enters the 230-km range from radar and lasted 24 h to capture its evolution before and after landfall. We present system demonstrations of two cases at different scales: a large-scale case of Hurricane Charley (2004) and a small-scale case of Tropical Storm Bill (2015). In the first case, Charley rapidly intensified prior to landfall in southwestern Florida and produced hurricane-force winds across the peninsula while also spawning nine tornadoes (Franklin et al. 2006). Charley contained deep convection and stratiform cloud regions (Matyas 2009) that required analysis at high spatial resolution in 3D to be rendered properly. As it also interacted with a frontal boundary across Georgia and North and South Carolina, the analysis of Charley required the merger of data from 17 radars. This case begins 0600 UTC 13 August 2004. We generated 3D mosaics every minute with a gridcell size of 0.1 km × 0.1 km × 0.1 km. Using 16 CPU cores on the University of Florida Research Computing (UFRC) cluster, the analysis of this test case was completed in about 15 h (Fig. 6).

Figures 3 and 4 show 2D and 3D snapshots of merged reflectivity data, and Fig. 3 shows a horizontal slice at 4-km altitude. The dashed line indicates a 230-km buffer zone from the center of WSR-88D radar stations included in this case. The inner-core structure is interpolated from echoes detected by radars located in Key West, Tampa, and Miami, Florida. This snapshot in the left inset clearly shows Charley’s broken eyewall before landfall as reported by Lee and Bell (2007). In Fig. 4, a 3D isosurface of reflectivity at 30 dBZ at
FIG. 3. Reflectivity contours of Hurricane Charley (2004) and frontal zone to the north at 2035 UTC 13 Aug. (top) Slice at a height of 4.0 km. “Contour” function in ArcGIS Runtime is employed to create polygons of contours at intervals of 2 dBZ. (bottom) Zoomed-in image of both reflectivity contours and the optimized horizontal wind vector near the eye.
the same time shows the double eyewall present during the analysis period discussed by Lee and Bell (2007). The inset on the right shows a zoomed-in view of reflectivity contour and horizontal wind field near the eye. The trend of wind direction shows the inner wind circulation of Charley is still closed. However, despite our best efforts of optimizing wind direction and speed, this result is still coarse. This is due to the drawback of VAD because VAD is not optimal when applied in tropical cyclone cases.

The second case of Tropical Storm Bill (2015) shows our system’s capability to handle both reflectivity contours and corresponding dual-polarization data. In this case, all three radars are dual polarimetric. The $\phi_{dp}$ is converted into $K_{dp}$ and then gridded with the same spatial resolution. Figure 5 shows spiral rainbands near the storm center after landfall at 0000 UTC 17 July 2015. Term $K_{dp}$ is expected to improve observation and identification of convective process and heavy rainfall area as compared to analysis of reflectivity (Li and Mecikalski 2012).

The Charley case is also used to test our system under a different hardware configuration to provide a comparison of performance when using 2–64 CPU cores (Fig. 6). As a general conclusion, one CPU core at 2.4 GHz with 4–6-GB memory digests 24 h of level II data from one radar station in 6–8 h. With increases in the number of CPUs and total memory, an approximated linear decrement of total running time is observed. However, this decrement is expected to slow or stop when the system reaches bottlenecks when caching data into the hard disk, exchanging large volumes of data between nodes and actors, and waiting for execution of geoprocessing functions in ArcGIS Runtime. To simulate these bottlenecks, a test was carried out on a low-end configured machine whose CPU runs about one-sixth as fast as one in the UFRC cluster, equipped with limited memory and less than 80 MB s$^{-1}$ hard disk.
FIG. 5. Reflectivity contours of Tropical Storm Bill (2015) at 0000 UTC 17 Jun. (top) Contour slice at a height of 4.0 km. (bottom) Corresponding mosaic of $K_{dp}$.
speed. The playback system took 70 h to run with identical outputs. Thus, we conclude that researchers with or without access to high-performance computing clusters could implement our playback system without concern for stability and scalability. Additionally, the performance is compared with Radx by running Radx to grid one radar station (KMLB; Melbourne, Florida) for the same time period with the same spatial resolution. On average, with 16 CPU cores of the UFRC cluster, Radx takes 1.6 h to create a single radar mosaic for one moment with the same spatial resolution. Thus, for a task of gridding total 17 radars, the estimated running time is about 27 h, which is slower. However, because of the functional limitations of Radx, which cannot temporally interpolate to any moments and is incompatible with the multiradar scenario, this comparison is not identical, but it shows a good acceleration of distributed parallelization provided by the playback system.

6. Summary

In this paper, a scalable, fast playback framework is presented, based on the MapReduce paradigm, for taking data (reflectivity and radial velocity) from multiple radars and combining them into a 3D merged domain. Level II radar data are mapped by time, elevation scan, and gates and then transformed into Cartesian coordinates in parallelized map functions in the format of RDDs. RDDs carrying key-value pairs are grouped, mosaicked into a 3D domain, and further interpolated with geospatial functions. We evaluated the scalability of the system and found it capable of running across both low-configuration single workstations and high-configuration clusters. It demonstrates good compatibility with traditional high-performance computing (HPC) architecture and job schedulers commonly available for advanced community and personal research. With a highly efficient in-memory data exchange protocol between Apache Spark and ArcGIS Runtime, the system successfully adopts GIS and geospatial functions into radar data processing that should enhance collaboration between researchers inside and outside the radar meteorology community.

This project presents our first effort to create an open-source, cross-platform, user-friendly project for largescale radar processing. Our future work will address solutions to the limitations that still exist. We will incorporate existing open-source projects, including the Python ARM Radar Toolkit (Helmus et al. 2013) and wradlib (Heistermann et al. 2013) as the preprocessing part in the playback system. This preprocessing component will provide functionality of supporting multiple data formats besides NEXRAD level II, as well as built-in quality control of inputs. Addressing the observed performance loss on Java-based code is critical. Also,
we plan to design a convenient graphical user interface (GUI) that lowers the requirements of professional training on radar processing and the Linux system for software operation. For example, users could work with the GUI on their Windows workstation, test and deploy geospatial functions with ArcGIS, and then execute and monitor jobs on clusters. A GUI-enabled environment will play a large role in filling the gap between the professional radar community and the use of radar data by researchers in other disciplines.

Acknowledgments. This work is supported by the National Science Foundation (BCS-1053864). We sincerely appreciate the existing innovative open-source radar projects for their inspiring algorithm designing. The comments from three anonymous reviewers significantly improved the manuscript.

REFERENCES


