arXiv:2101.01852v1 [cs.DB] 6 Jan 2021

Bridging BAD Islands: Declarative Data Sharing at Scale

Xikui Wang, Michael J. Carey
Donald Bren School of Information and Computer Sciences
University of California Irvine
Irvine, United States
{xikuiw, mjcarey}@ics.uci.edu

Vassilis J. Tsotras
Department of Computer Science and Engineering
University of California Riverside
Riverside, United States
tsotras@cs.ucr.edu

***Abstract*—In many Big Data applications today, information needs to be actively shared between systems managed by different organizations. To enable sharing Big Data at scale, developers would have to create dedicated server programs and glue together multiple Big Data systems for scalability. Developing and managing such glued data sharing services requires a significant amount of work from developers. In our prior work, we developed a Big Active Data (BAD) system for enabling Big Data subscriptions and analytics with millions of subscribers. Based on that, we introduce a new mechanism for enabling the sharing of Big Data at scale *declaratively* so that developers can easily create and provide data sharing services using declarative statements and can benefit from an underlying scalable infrastructure. We show our implementation on top of the BAD system, explain the data sharing data flow among multiple systems, and present a prototype system with experimental results.**

***Index Terms*—data warehouses, database systems, distributed information systems**

I. Introduction

Advances in information technology have created large collections of data [1]. Such large volumes of data - Big Data - also come with big challenges. In order to transmit, process, and persist Big Data, researchers and experts from academia and industry have developed a plethora of systems [2]–[5]. However, most of them are passive in nature - passively answering users' requests to process and return data rather than actively processing and delivering data of interest to users. In many applications, users not only want to analyze data, but also to subscribe to and actively receive data of interest. Their interests may include the data's content as well as its relationships to other data. For example, in-field officers may want to *receive **nearby** threatening tweets **whenever they are posted***. There can be millions of users having similar requests. We refer to the enabling of Big Data subscriptions and analytics as *Big Active Data* (BAD). Traditional pub/sub systems [6] often lack the capability of data processing and handling complex subscription requests that involve data's relationships (*such as send me tweets **near my current location***). More recent stream processing engines [7], [8] usually don't persist data for historical data analytics (*such as show me the average threatening rating of tweets in the **past five months** grouped by their location*). In order to accommodate BAD challenges, we have created a BAD system that supports Big Data subscriptions and analytics at scale [9]–[13].

In a big BAD world, the data to be analyzed and delivered often needs to be processed and enriched with additional information so that interested users can obtain more insights from the data. Such additional information may be managed by different organizations. Developers often need to share data between different systems for supporting BAD applications (e.g., threatening tweets detected at the Department of Homeland Security need to be shared with local police departments). Data sharing can be difficult, besides the ethical and legal issues, because of the challenges in management, interoperability, security, and infrastructure [14]. Researchers from academia have developed projects that unify institutional repositories from different organizations for sharing research datasets [15], [16]. Companies have also created platforms based on Big Data projects to improve business efficiency and consolidate resources for better services [17]. Nevertheless, providing efficient, reliable, and scalable data sharing services require dedicated infrastructures and collaborative efforts from developers and organizations. In this work, we focus on enabling the active sharing of Big Data declaratively in a BAD world. In particular, we characterize a BAD world as a group of BAD islands, where each organization runs an independent BAD system as an island. We discuss how to "bridge" different BAD islands using scalable data sharing services without additional programming from developers.

II. Big Active Data in a Nutshell

Our BAD system has been built as an extension of Apache AsterixDB, a Big Data Management System (BDMS) that provides distributed data management for large-scale, semi-structured data [18]. The BAD system [10], [13] can enable millions of users to subscribe to data of interest and receive updates continuously, and it also supports Big Data analytics with a declarative language, SQL++ (a SQL-inspired query language for semi-structured data) [19]. An overview of the BAD system is shown in Figure 1. Due to space limits, here we focus on two key components: Data Feeds and Data Channels. For more details about our project we refer to [9]–[13].

978-1-7281-6251-5/20/$31.00 ©2020 IEEE

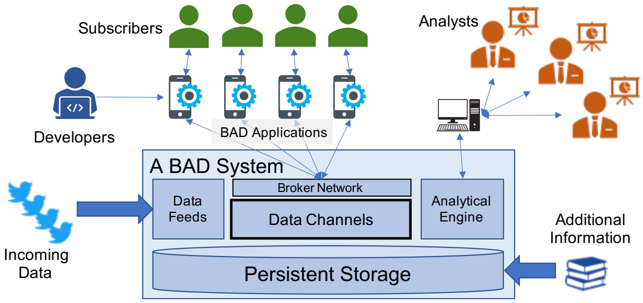

Fig. 1. An overview of the BAD system

A. Data Feeds

Data feeds help the BAD system to ingest rapidly incoming data from various external data sources in different formats. Users can create a data feed using SQL++ statements. As an example, in Figure 2, we define a data type *Tweet* to describe the incoming data's minimum required attributes and an active dataset *Tweets* to persist the incoming data. Active datasets, different from normal datasets, enable continuous query semantics [20] in channels (discussed next) [10], [12]. Here we create a *TweetFeed* using a socket adapter and specify the incoming data's format as JSON. This allows the BAD system to use a socket server to intake incoming JSON data. The *TweetFeed* is connected to the dataset *Tweets* so that the ingested data can be persisted in storage (partitioned across all nodes of a cluster) directly for later use. There are two types of data feeds: static feeds, which maximize ingestion throughput, and dynamic feeds, which allow users to enrich incoming data using user-defined functions (UDFs) [21]–[24].

```
CREATE TYPE Tweet AS { tid: bigint, uid: bigint, text: string };
CREATE ACTIVE DATASET Tweets(Tweet) PRIMARY KEY tid;
CREATE FEED TweetFeed WITH {
  "type-name" : "TweetType",
  "adapter-name": "socket_adapter",
  "format" : "JSON",
  "sockets": "FEED_HOST:FEED_PORT",
  "address-type": "IP",
  "dynamic": false };
CONNECT FEED TweetFeed TO DATASET Tweets;
START FEED TweetFeed;
```

Fig. 2. A sample data feed connected to an active dataset

B. Data Channels

Data channels allow developers to activate parameterized queries as services for millions of users to subscribe to and continuously receive their data of interest. When creating a channel, developers can construct a *channel query* to describe the data of interest for subscribers and specify a *channel period* to indicate how often should the channel query be evaluated for subscribed users. All subscriptions of a channel are evaluated together to allow the system to exploit shared computations among them (e.g., many subscribers could be interested in tweets from Orange County), and increasing the channel period could lead to a bigger batch size and thus allow computing complex data of interest for more subscribers at scale. For example, we can create a *NearbyThreateningTweets* channel, as shown in Figure 3, to allow in-field officers to subscribe to nearby threatening tweets. In the channel query, we use the *is_new* function to look for **new** threatening tweets near a subscribed officer's location and return those tweets to subscribers every 10 seconds.[1] The active dataset *Tweets* provides continuous query semantics to make sure every qualified new tweet will be delivered to subscribed officers. The threatening tweets for subscribers are sent to brokers registered as HTTP endpoints in the BAD system. A user can subscribe to a data channel on a broker and thus receive updates from it. As shown in Figure 4, we can register a broker and make two separate subscriptions (on behalf of in-field officers) on this broker so that the threatening tweets near these two in-field officers are sent to this broker and then delivered to them. Data channels provide two modes for delivering data: *push* and *pull*. In the push mode, the data of interest is pushed to brokers directly. In the pull mode, a broker having new data of interest for its subscribers will receive a notification from the channel, and then the broker can pull that data from BAD storage later.

```
// Similar to TweetFeed and Tweets, we have a LocationFeed connected
// to an OfficerLocations dataset to receive and store the live
// location updates from in-field officers
// CREATE TYPE OfficerLocation AS { oid: int, location: point };
// CREATE ACTIVE DATASET OfficerLocations(OfficerLocation)
//   PRIMARY KEY oid;

CREATE CONTINUOUS CHANNEL NearbyThreateningTweets(oid)
 PERIOD duration("PT10S") {
  SELECT t FROM OfficerLocations o, Tweets t
  WHERE spatial_distance(t.location, o.location) < 5
    AND o.oid = oid AND t.threatening_rating > 0 AND is_new(t) };
```

Fig. 3. A sample continuous channel for nearby hateful tweets

```
CREATE BROKER BROKER_A AT "http://BROKER_A_HOST:BROKER_A_PORT/API";
SUBSCRIBE TO NearbyThreateningTweets("0907") ON BROKER_A;
SUBSCRIBE TO NearbyThreateningTweets("1226") ON BROKER_A;
```

Fig. 4. Registering a broker and making subscriptions

III. BAD Islands

In the following sections, we discuss how we can connect BAD systems managed by different organizations (islands) in a BAD world together to enable data sharing among them. We use a three-island example with the following organizations for illustration: the Department of Homeland Security, the Orange County Sheriff's Department, and the University of California-Irvine. Each organization hosts an independent BAD system and serves its own BAD users with localized information.

A. BAD Island 1: Department of Homeland Security

The Department of Homeland Security (DHS) is a federal agency responsible for ensuring public security. In our example, DHS has access to all tweets posted in the United States. These tweets cannot be shared with other organizations directly due to licensing and privacy concerns, except for the tweets that are related to potential threats. The BAD system at DHS needs to provide data analytics on collected tweets and serve tweets to its agents through data channels.

Since raw tweets from Twitter may not contain all necessary information, DHS might need to enrich them with other

[1] One could also apply the *is_new* function on OfficerLocations to look for nearby threatening tweets only for officers actively updating their locations. Interested readers may refer to [13] for more continuous channel examples.

relevant data. As an example, DHS could collect weapon registration information for some sensitive twitter account holders and attach that to tweets to provide important additional information for interested subscribers. In addition, DHS could also utilize Machine Learning algorithms to estimate the threatening rating of the tweets' text and use that for later analysis. An overview of the DHS island is shown in Figure 5.

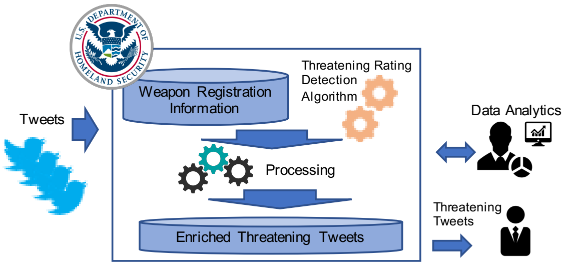
Tweets
Weapon Registration Information
Threatening Rating Detection Algorithm
Processing
Enriched Threatening Tweets
Data Analytics
Threatening Tweets

Fig. 5. An overview of the DHS Island

B. BAD Island 2: Orange County Sheriff's Department

The Orange County Sheriff's Department (OCSD) is the local law enforcement agency that ensures safety and responds to potential crimes in Orange County, CA. In our use case, OCSD wants to monitor major local events and ensure the safety of the event and its participants. In-field officers who patrol around the county, continuously report their locations back to OCSD so that OCSD can send them instructions based on their locations (e.g., when an emergency happens, send nearby officers for help).

To prevent potential threats to local events, OCSD would like to obtain the threatening tweets posted in Orange County. When a local threatening tweet is detected, OCSD can find important events close to the tweet and then notify the nearby in-field officers about the event and the tweet so they can further investigate it. Additionally, OCSD wants to support data analytics on data stored in the system. An overview of the OCSD island is shown in Figure 6.

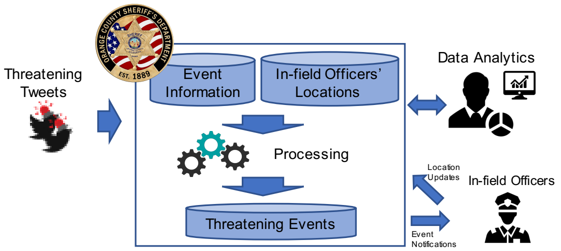

Fig. 6. An overview of the OCSD island

C. BAD Island 3: University of California-Irvine

The University of California-Irvine (UCI) is a public university located in Irvine, a city in Orange County. The university often hosts various activities and events in different buildings on campus. To ensure students' and visitors' safety, the university has its own university police officers placed at various security stations on campus, and students/visitors can seek help from when an emergency happens. The buildings on campus have notice boards for showing important notifications and alerts. The university also has an alerting service - zotALERT - which delivers important messages to people (subscribers) on-campus through text messages and emails.

UCI would like to acquire the threatening tweets posted near the UCI campus and notify people in the buildings around those tweets to raise attention. An alert could include the information about nearby security stations for the tweet so that people in an emergency situation could quickly seek help. Data analytics on threatening tweets and other data in the system for school officials are also to be supported. An overview of the UCI island is shown in Figure 7.

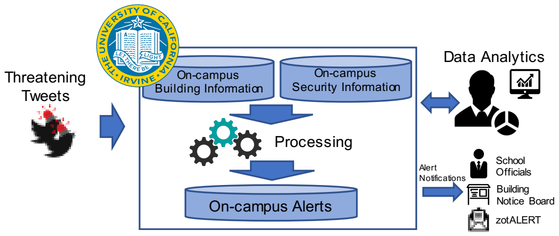

Fig. 7. An overview of the UCI island

IV. Island Hopping: Connecting BAD Islands

In order to support the BAD services at OCSD and UCI described in Section III, we need to enable the sharing of threatening tweets detected at DHS with OCSD and UCI. These tweets can be combined with local information at Orange County and UCI, respectively, and then be used for creating localized notifications for subscribers on each island. Below we consider three options for sharing threatening tweets among these islands, namely: (1) combining all islands together into one (*a BAD Continent*), (2) creating direct connections between the individual islands as needed (*BAD Ferries*) and (3) utilizing the channel idea to allow islands to subscribe to what they need from one another (*BAD Bridges*). Below we discuss the three options in detail.

A. Option 1: A BAD Continent

Instead of sharing threatening tweets between multiple BAD islands, one could create a big BAD island, namely a BAD continent, that holds not only the data at DHS but also the local data from OCSD and UCI, as shown in Figure 8. In this case, all services at OCSD and UCI could be integrated into this BAD continent, and all subscribers then would subscribe to this BAD continent directly. All information is now in the same system. Developers from different organizations could easily create BAD services without having to share data.

In principle, a one-for-all BAD continent could be easy to build, and it avoids the complexity of connecting different BAD islands. Although the resulting BAD system could be scaled to support the volume of data and users from multiple organizations, such global integration would introduce significant management and administration overheads, especially for the service provider (DHS in this case). For the three-island example, not only would all local information (including local events, campus building layouts, etc.) need to be stored in the BAD continent, but all updates (location updates, event updates, etc.) would need to be forwarded to the system. Managing all local data at DHS could be very complex

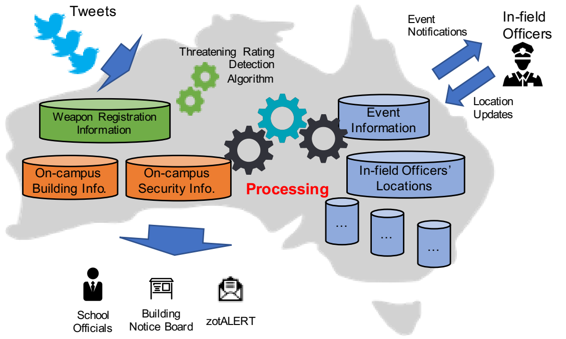

Fig. 8. An illustration of a BAD continent

and would require sophisticated access control. When more organizations join, such a database would have to manage all kinds of additional local information while receiving updates from multiple parties; this system would quickly become impractical to maintain by one organization. Additionally, such global information sharing may not be permitted (by law) between different agencies in all cases.

B. Option 2: BAD Ferries

A different way of supporting the required BAD services at OCSD and UCI, without combining everything together, would be to programmatically send the requested data from DHS to OCSD and UCI, as shown in Figure 9. DHS could send the threatening tweets detected in Orange County and near UCI campus to OCSD and UCI, respectively, and OCSD BAD and UCI BAD could then combine those tweets with their local information to produce localized notifications for their subscribers.

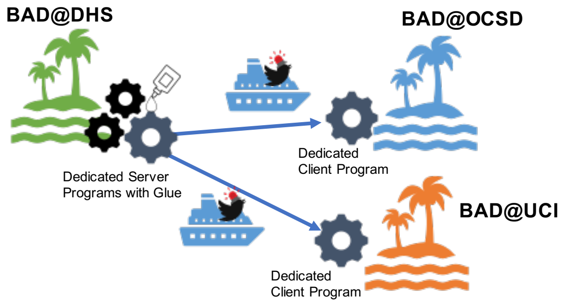

Fig. 9. An illustration of BAD ferries

In order to share the data cleanly and efficiently, DHS would need to create a dedicated server program that allows other organizations to access the shared data in DHS. Also, OCSD and UCI would need to develop corresponding client programs connected to the DHS server program and obtain shared data. Data exchanges between the server and clients could be frequent, and there could be many more clients who would like to access the shared data. Thus, the server program would need to be efficient, reliable, and scalable for handling a large number of clients and a large volume of data. Implementing and extending the server and client programs would require significant efforts from these organizations.

C. Option 3: BAD Bridges

An important observation is that this data exchange pattern, where we have an island serving data and multiple islands constantly requesting data of interest, resonates well with the original BAD user model, where subscribers subscribe to data and constantly receive updates. Inspired by this, we could characterize a BAD island as being a BAD subscriber of another island and connect these islands using *BAD bridges* built on data channels and data feeds to share data at scale, as shown in Figure 10. One might characterize this architecture as: *"One man's channel is another man's feed."*

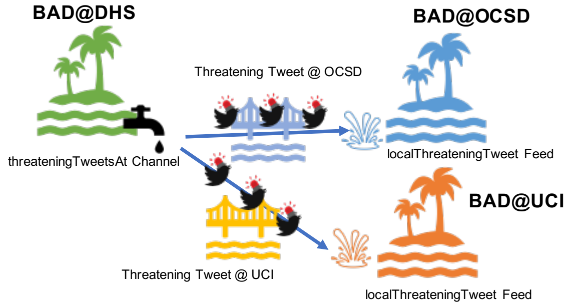
BAD@DHS
BAD@OCSD
Threatening Tweet @ OCSD
threateningTweetsAt Channel
localThreateningTweet Feed
BAD@UCI
Threatening Tweet @ UCI
localThreateningTweet Feed

Fig. 10. An illustration of BAD bridges

Following our example, we could first create a data channel on DHS BAD, which serves threatening tweets by areas, namely via a threateningTweetsAt channel, and other islands interested in local threatening tweets from an area could then subscribe to this channel with the area name of interest. OCSD BAD, as a subscriber, can subscribe to this channel with the parameter "OC", and UCI BAD, as another subscriber, can also subscribe to this channel with the parameter "UCI". We could use a push channel to push threatening tweets to OCSD and UCI BAD so they can receive local threatening tweets from the channel at DHS directly, process them with local information, and then produce localized notifications to their own subscribers.

On OCSD and UCI BAD, we could utilize data feeds to receive threatening tweets detected by the threateningTweetsAt channel on DHS BAD. Taking OCSD BAD as an example, we could create an HTTP feed and connect it to a local OCSD dataset for persisting the threatening tweets. We could register the feed's HTTP address as a broker in DHS BAD and then subscribe to the threateningTweetsAt channel with the parameter "OC". With this feed, broker, and subscription, threatening tweets posted at Orange County and detected by DHS would then be sent to the feed's endpoint from the threateningTweetsAt channel. Similarly, we could repeat this process for other BAD systems to obtain threatening tweets from their areas of interest. Since the BAD system is scalable and can support a large number of subscribers with a large volume of data, bridging BAD systems using data channels and feeds can be scaled out to support many more islands connecting to DHS. This allows developers to declaratively create data sharing services, without additional programming and gluing together multiple systems, as we will see next.

V. Building BAD Bridges

Given the advantages of the BAD Bridges approach, we now introduce *BAD brokers* to further simplify and enhance data exchanges between BAD islands and *BAD feeds* and thus help users create bridges and manage their life-cycles.

A. BAD Brokers

The broker sub-system in BAD manages the communication between the BAD system and its subscribers. A broker registers itself as an HTTP endpoint in the BAD system. Notifications containing data of interest produced by the BAD backend are delivered to this broker endpoint and then disseminated to subscribers who subscribed on this broker. In order to allow general brokers to parse the incoming notifications, data channels produce notifications as JSON objects, and more complex data types supported in BAD in the AsterixDB Data Model (ADM) (such as datetimes, points, etc.) are encoded as strings, arrays, and other JSON data types. Since BAD islands are "brokers" that can also directly process ADM data, we can instead deliver their notifications as ADM records to maintain the richer data type information and avoid additional data encoding and decoding overheads.

To allow brokers to process ADM data and to become extensible for future use cases, we introduce a new notion of *BAD brokers* and a simple new syntax for creating brokers in BAD, as shown in Figure 11. Users can add an optional WITH statement for providing additional information about the broker. While we only support "broker-type" for now, this can be further extended to support other features in the future. When there is no WITH statement or when the broker-type is set to "general", we create a general broker that takes JSON data. When the broker-type is set to "BAD", we create a BAD broker that takes ADM records. In general, a channel can have subscriptions from both types of brokers. In that case, channel executions will send JSON formatted data to the general brokers and ADM formatted data to the BAD brokers.

```
CREATE BROKER BROKER_NAME AT "http://BROKER_HOST:PORT_NUM" WITH
  { "broker-type" : "BAD" };
```

Fig. 11. Creating a BAD broker

B. BAD Feeds

Bridging from a BAD island *A* to another BAD island *B* and sharing data to island *B* requires several steps: create a data feed on island *B*; register the feed with island *A* as a BAD broker; and create a subscription on island *A* on the created broker. Also, removing the bridge between island *A* and island *B* requires unsubscribing from the channel and removing the BAD broker on island *A*. In order to simplify the process of bridging BAD islands and help users manage the life-cycles of bridges, we also introduce the notion of *BAD feeds*.

One can create a BAD feed on island *B* and connect it to a channel on island *A* using the statement in Figure 12. Unlike regular data feeds, users would need to specify several additional configuration entries for connecting to a data channel on the other BAD island. In particular, the "bad-host", "bad-channel", and "bad-dataverse" configuration parameters help the system locate the data channel on the other island, while "bad-channel-parameters" contains subscription parameters as a quote-escaped string for subscribing to the channel. When a channel takes multiple parameters, we use commas to separate them. If a data feed wants to subscribe to a channel with several different parameters (e.g., OCSD BAD wants to subscribe to threatening tweets from both Orange County and UCI) we can concatenate them using semicolons.

```
CREATE FEED A_SAMPLE_BAD_FEED_ON_ISLAND_B WITH {
  "adapter-name" : "http_adapter",
  "address-type" : "IP",
  "format" : "ADM",
  "addresses" : "ISLAND_B_FEED_HOST:ISLAND_B_FEED_PORT",
  "type-name" : "INCOMING_DATA_TYPE",
  "bad-host" : "ISLAND_A_HOST",
  "bad-channel" : "ISLAND_A_CHANNEL_NAME",
  "bad-channel-parameters": "PARAM_1-1,PARAM_1-2;PARAM2-1,PARAM_2-2",
  "bad-dataverse": "ISLAND_A_CHANNEL_DATAVERSE" };
```

Fig. 12. Creating a BAD feed on island B

The bridge's information is persisted in the BAD system's metadata with a feed's configuration when the feed is created. When a user starts a BAD feed on a local BAD system (island *B*), it registers a broker on the specified remote BAD system (island *A*) using island B's feed endpoint and subscribes to island A's channel using the provided parameters automatically. When a user stops the BAD feed, island *B* unsubscribes from island A's channel and then removes the broker from island *A*. We tie the start and stop events of a data feed on the local BAD system (island B) to the subscribe and unsubscribe actions on the remote BAD system (island A), so when the feed is not running, the remote BAD system will not need to compute and deliver data to this BAD feed.

VI. A Prototype of BAD Islands

We now describe a complete prototype of BAD islands that supports the use cases described in Section III. We show how to create and connect three BAD islands (the BAD trinity) using declarative statements and show how data flows between these different islands. The BAD system organizes data and other entities under *dataverses* (similar to databases in an RDBMS). To differentiate organizations, we use different dataverses for different organizations (using the **USE** statement).

A. BAD@DHS

DHS BAD intakes tweets from external data sources. We can create a TweetFeed like the one in Figure 2 and configure it as dynamic to enrich the incoming tweets with additional information needed using UDFs. We first enrich an incoming tweet with the tweet's user's weapon registration records (if any). To hold the weapon registration records of sensitive tweet users, we create a data type *WeaponRegistration* and a dataset *WeaponRegistrations*, as shown in Figure 13. (A user may have multiple weapons.)

```
USE dhs;
CREATE TYPE WeaponRegistration AS
  { wrid: uuid, uid: bigint, weapon_name: string };
CREATE DATASET WeaponRegistrations(WeaponRegistration)
  PRIMARY KEY wrid AUTOGENERATED;
```

Fig. 13. Data type and dataset definition for weapon registration information

Second, we create a Java UDF to detect the threatening rating of a tweet's text using a list of threatening words, as shown in Figure 14. In this UDF, we load an external list of threatening words and we use the number of threatening words in the given text as its threatening rating.

```
...
@Override
public void evaluate(IFunctionHelper functionHelper) throws Exception {
    JString input = (JString) functionHelper.getArgument(0);
    JInt output = (JInt) functionHelper.getResultObject();
    String tweetText = input.getValue();
    int threateningRating = 0;
    String[] words = tweetText.split(" ");
    for (String word : words) {
        // The threateningWordList is initialized with a file when function starts
        if (threateningWordList.contains(word.replaceAll("[,.]", ""))) {
            threateningRating++;
        }
    }
    output.setValue(threateningRating);
    functionHelper.setResult(output);
}
...
```

Fig. 14. A Java UDF for determining the threatening rating

To add the desired set of enrichments to incoming tweets, we can create a SQL++ UDF *EnrichTweet* and attach it to the TweetFeed when connecting to the Tweets dataset, as shown in Figure 15. In this UDF, we also transform the epoch time of a tweet's "created_at" attribute into a datetime attribute "timestamp" and we create a point attribute "location" using the array of coordinates. These ADM attributes can be useful, as they do not need to be constructed in computations like spatial joins every time. Here we use the Java UDF defined in Figure 14 to extract the threatening rating of the tweet's text and attach it as a "threatening_rating" attribute. We use a sub-query to look for the weapon registration information of the tweet's user and nest the registered weapons into a "user_registered_weapon" attribute. These new attributes are merged into the tweet and will be persisted for producing notifications.

```
USE dhs;
CREATE FUNCTION EnrichTweet(tweet) {
  object_merge(tweet, {
    "timestamp" : datetime_from_unix_time_in_ms(tweet.created_at),
    "location" :
       create_point(tweet.coordinates[0],tweet.coordinates[1]),
    "threatening_rating" : threateningRating(tweet.text),
    "user_registered_weapon": (SELECT VALUE w.weapon_name
       FROM WeaponRegistrations w WHERE w.uid = tweet.uid)})
};
CONNECT FEED TweetFeed to DATASET Tweets APPLY FUNCTION EnrichTweet;
START FEED TweetFeed;
```

Fig. 15. Enriching tweets with additional information

With these enriched threatening tweets, we can serve threatening tweets from areas by creating the continuous data channel "ThreateningTweetsAt" shown in Figure 16. To put everything together, a detailed overview of the entire DHS BAD system is shown in Figure 17.

```
USE dhs;
CREATE CONTINUOUS PUSH CHANNEL ThreateningTweetsAt(area_name)
 PERIOD duration("PERIOD_DURATION") {
  SELECT t.area_name, t.text, t.location, t.threatening_rating,
    t.user_registered_weapon FROM Tweets t
  WHERE t.area_name = area_name
    AND t.threatening_rating > 0 AND is_new(t) };
```

Fig. 16. Definition of the ThreateningTweetsAt channel

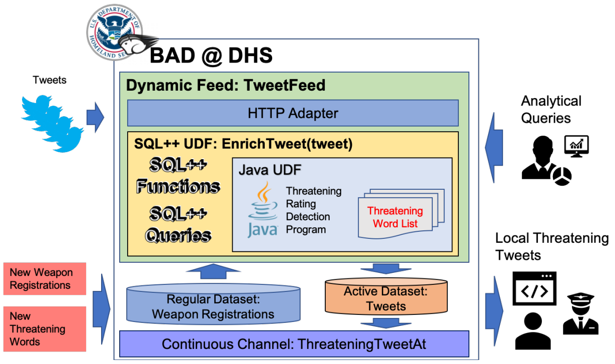

Fig. 17. The internal details of the DHS BAD system

B. BAD@OCSD

OCSD BAD in this prototype receives threatening tweets not only from Orange County but also from UCI to demonstrate how a BAD feed can connect to a channel with two sets of parameters. OCSD BAD notifies in-field officers about nearby threatening tweets that are close to important local events. To persist event information in OCSD BAD, we can create a data type "Event" and a dataset "Events" as shown in Figure 18.

```
USE ocsd;
CREATE TYPE Event AS { eid: uuid, name: string, location: point,
  event_duration: duration, radius_km: double };
CREATE DATASET Events(Event) PRIMARY KEY eid;
```

Fig. 18. Data type and dataset definition for events

To store threatening tweets coming from DHS BAD, we create a data type *LocalThreateningTweet* and an active dataset *LocalThreateningTweets* in Figure 19. (We use an active dataset for threatening tweets to ensure continuous query semantics in later local channel computations.) We create a BAD feed in Figure 20 to obtain local threatening tweets from DHS. This BAD feed subscribes to the DHS threateningTweetsAt channel with parameters "OC" and "UCI", which correspond to two separate subscriptions in DHS BAD. Since there is no further data enrichment during ingestion, we use a static data feed here and connect it to LocalThreateningTweets directly.

```
CREATE TYPE LocalThreateningTweet AS
  { channelExecutionEpochTime: bigint,
    dataverseName: string, channelName: string };
CREATE ACTIVE DATASET LocalThreateningTweets(LocalThreateningTweet)
 PRIMARY KEY channelExecutionEpochTime;
```

Fig. 19. Data type and dataset definition for local threatening tweets at Orange County

```
USE ocsd;
CREATE FEED LocalThreateningTweetFeed WITH {
  "adapter-name" : "http_adapter",
  "addresses" : "OCSD_HOST:10013",
  "address-type" : "IP",
  "type-name" : "LocalThreateningTweet",
  "format" : "adm",
  "bad-host" : "DHS_HOST",
  "bad-channel" : "ThreateningTweetsAt",
  "bad-channel-parameters": "\"OC\";\"UCI\"",
  "bad-dataverse": "dhs",
  "dynamic": false };
CONNECT FEED LocalThreateningTweetFeed
  TO DATASET LocalThreateningTweets;
START FEED LocalThreateningTweetFeed;
```

Fig. 20. Definition, connect and start feed statements for LocalThreateningTweetFeed

In-field officers from OCSD also continuously send their location updates to the OCSD BAD system so that OCSD can notify the officers about nearby threatening tweets based on their current location. We can use the data type, dataset, and feed described in Figure 3 for intaking and persisting the location updates. As there is no further enrichment for location updates, the LocationFeed can be static as well.

With the local threatening tweets, event information, and officers' locations, we can now create a continuous channel for in-field officers to subscribe to nearby threatening tweets close to local events (a.k.a. threatening events), as shown in Figure 21. The notifications from DHS contain threatening tweets as an array in the "results" attribute, so we use the UNNEST operation to access each independent threatening tweet. We calculate the distance between the officer and the tweet, the event and the tweet, and the officer and the event. If the officer is near a threatening tweet and the threatening tweet is near an event, we send a notification to the officer. The notification contains the tweet's content, the event information, the distance between the officer and the tweet, and the distance between the officer and the event in the notification to help the officer take further actions. A detailed overview of the OCSD BAD system is shown in Figure 22.

```
USE ocsd;
CREATE CONTINUOUS PUSH CHANNEL ThreateningEventsNear(oid)
 PERIOD duration("PERIOD_DURATION") {
  FROM LocalThreateningTweets tn, OfficerLocations o, Events e
  UNNEST tn.results threatening_tweet
  LET tweet_loc = threatening_tweet.result.location,
  officer_tweet_dist = spatial_distance(o.location, tweet_loc),
  event_tweet_dist = spatial_distance(e.location, tweet_loc),
  officer_event_dist = spatial_distance(o.location, e.location)
    WHERE is_new(tn) AND oid = o.oid AND officer_tweet_dist < 0.1
      AND event_tweet_dist < e.radius_km / 100
  SELECT oid, threatening_tweet.result tweet_content, e event_info,
    officer_tweet_dist * 100 as tweet_distance_km,
    officer_event_dist * 100 as event_distance_km
};
```

Fig. 21. Definition of the ThreateningEventsNear channel

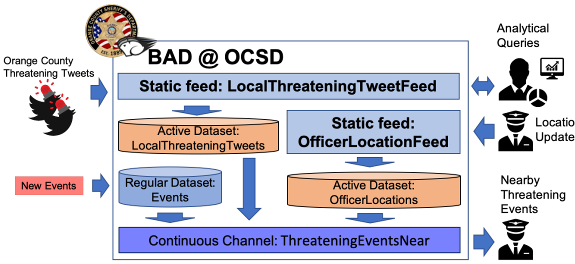
Orange County Threatening Tweets
New Events
BAD @ OCSD
Static feed: LocalThreateningTweetFeed
Active Dataset: LocalThreateningTweets
Static feed: OfficerLocationFeed
Regular Dataset: Events
Active Dataset: OfficerLocations
Continuous Channel: ThreateningEventsNear
Analytical Queries
Location Updates
Nearby Threatening Events

Fig. 22. The internal details of the OCSD BAD system

C. BAD@UCI

UCI BAD receives threatening tweets posted at UCI and checks whether a threatening tweet is near an on-campus building. If so, it creates a notification about the threatening tweet together with the nearby security stations' information. Like OCSD BAD, to persist threatening tweets at UCI, we need to create a data type *LocalThreateningTweet* and a dataset *LocalThreateningTweet* on UCI BAD. To receive threatening tweets at UCI from DHS, we need to create a BAD feed, like Figure 20, connected to the ThreateningTweetsAt channel but using the parameter "UCI".

To provide more information for UCI BAD's subscribers, we store on-campus buildings, for checking whether there is a threatening tweet nearby, and security stations, for students to seek for help from, in UCI BAD. In Figure 23, we create the data types and datasets for them respectively.

```
USE uci;
CREATE TYPE Building AS { bid: uuid, name: string };
CREATE TYPE SecurityStation AS { sid: bigint, location: point };
CREATE DATASET Buildings(Building) PRIMARY KEY bid AUTOGENERATED;
CREATE DATASET SecurityStations(SecurityStation) PRIMARY KEY sid;
```

Fig. 23. Data type and dataset definition of buildings and security stations

With the local threatening tweets, on-campus building information, and security station information, we can create a continuous channel called "AlertsOnCampus" to provide on-campus alerts about threatening tweets near buildings with security stations' information attached using the statement shown in Figure 24. Like the ThreateningEventsNear channel in OCSD BAD, we first UNNEST threatening tweets from the incoming notifications. Then, we check whether a threatening tweet is posted at an on-campus building. If so, we attach the security station information to the threatening tweet, with stations ordered by their distances to the tweet's location, and generate an alert. A detailed overview of the UCI BAD system is shown in Figure 25.

```
USE uci;
CREATE CONTINUOUS PUSH CHANNEL AlertsOnCampus()
 PERIOD duration("PERIOD_DURATION") {
  FROM LocalThreateningTweets tn, Buildings b
  UNNEST tn.results threatening_tweet
  LET tweet_loc = threatening_tweet.result.location,
    station_dist = (FROM SecurityStations s
      LET dist = spatial_distance(tweet_loc, s.location)
      SELECT s stationInfo, dist * 100 dist_km ORDER BY dist)
  WHERE is_new(tn) AND spatial_intersect(tweet_loc, b.area)
  SELECT threateningTweet.result tweet_content,
    b building_info, station_dist
};
```

Fig. 24. Definition of the AlertsOnCampus channel

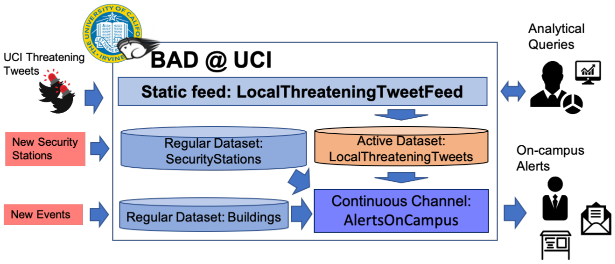
UCI Threatening Tweets
New Security Stations
New Events
BAD @ UCI
Static feed: LocalThreateningTweetFeed
Regular Dataset: SecurityStations
Active Dataset: LocalThreateningTweets
Regular Dataset: Buildings
Continuous Channel: AlertsOnCampus
Analytical Queries
On-campus Alerts

Fig. 25. The internal details of the UCI BAD system

D. The Trip of A Threatening Tweet

In order to illustrate how BAD islands interact with BAD bridges, we pick a sample tweet and show how it flows through the three islands and their bridges and produces notifications with local information for the subscribers on each island. An overview of our three-island prototype is shown in Figure 26. The circled numerical labels in the figure will be used later for illustrating the data content at different stages of the workflow.

We will use the raw tweet in Figure 27 (labeled 1 in Figure 26) as the example. This tweet is posted at UCI, and it contains the tweet's geolocation as a JSON array of coordinates and the epoch timestamp of when the tweet was created as a JSON number. This raw tweet is ingested by the TweetFeed defined in Figure 2 and then enriched by the UDF defined in Figure 15. After that, the enriched tweet is persisted in the Tweets dataset as shown in Figure 28 (labeled 2 in Figure 26). Enriched tweets contain a threatening rating detected

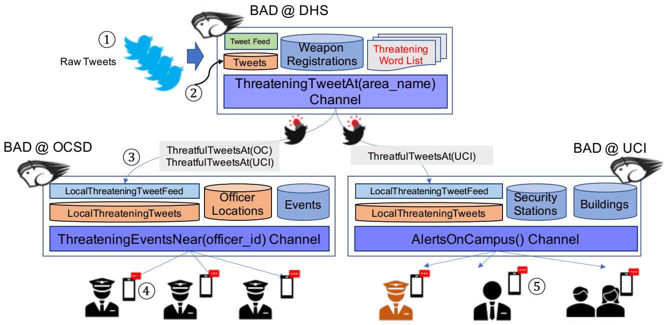

Fig. 26. An overview of BAD islands

```
{
  "tid": 1593142018123,
  "uid": 73,
  "area_name": "UCI",
  "text": "Saul Goodman builds SKS, and Todd Alquist fires AK47,
           but Skyler White sells Cabbage.",
  "coordinates": [ 33.64921228736088, -117.84181977473024 ],
  "created_at": 1593142018123
}
```

Fig. 27. A sample raw threatening tweet

```
{
  "tid": 1593142018123,
  "uid": 73,
  "area_name": "UCI",
  "text": "Saul Goodman builds SKS, and Todd Alquist fires AK47,
           but Skyler White sells Cabbage.",
  "coordinates": [ 33.64921228736088, -117.84181977473024 ],
  "created_at": 1593142018123,
  "threatening_rating": 2,
  "user_registered_weapon": [ "AR10" , "AK47", "GLOCK21" ],
  "timestamp": datetime("2020-06-26T03:26:58.123Z"),
  "location": point("33.64921228736088,-117.84181977473024")
}
```

Fig. 28. The enriched threatening tweet

by the Java UDF, an array of registered weapons for the tweet's user obtained by looking in the WeaponRegistrations dataset using the "uid" attribute, a timestamp as a datetime attribute, and a location as a point attribute.

Since both the OCSD and UCI BAD systems subscribe to threatening tweets at UCI, they will each receive a notification from DHS BAD about this threatening tweet. Figure 29 shows the notification sent to OCSD BAD (labeled 3 in Figure 26). If there was also a threatening tweet posted in Orange County at the same time, the "results" array would include that tweet but with a different subscription ID, as OCSD BAD has two subscriptions to the ThreateningTweetAt channel with parameters "OC" and "UCI", respectively. Since UCI BAD also subscribes to the channel, but with a different subscription on another broker (pointed to UCI's BAD feed), the notification for UCI BAD will be produced and sent separately.

In the OCSD BAD ThreateningEventsNear channel, threatening tweets are combined with local event information and officer location information to produce the nearby threatening event notifications for in-field officers. There is one local event "OC Marathon" near the threatening tweet in Figure 28, and there is an in-field officer 0 nearby, so OCSD BAD produces one notification about the tweet and the event for this officer. Figure 30 shows this threatening event notification (labeled 4 in Figure 26). It contains the event information as the "event_info" attribute, the threatening tweet's information as the "tweet_content" attribute, and the distances from the officer 0 to the tweet and to the event as the "event_distance_km" and "tweet_distance_km" attributes respectively.

```
{
  "dataverseName": "dhs",
  "channelName": "ThreateningTweetsAt",
  "channelExecutionEpochTime": 1593142019521,
  "results": [
    {
      "result": {
        "text": "Saul Goodman builds SKS, and Todd Alquist fires
                AK47, but Skyler White sells Cabbage.",
        "area_name": "UCI",
        "location": point("33.64921228736088,-117.84181977473024"),
        "threatening_rating": 2,
        "user_registered_weapon": [ "AR10" , "AK47", "GLOCK21" ]
      },
      "channelExecutionTime": datetime("2020-06-26T03:26:59.521Z"),
      "subscriptionId": uuid("82e61d25-f7ad-0632-3b9a-9c26e681ad84"),
      "deliveryTime": datetime("2020-06-26T03:26:59.522Z")
    }
  ]
}
```

Fig. 29. The generated threatening tweet notification from DHS

```
{
  "dataverseName": "ocsd",
  "channelName": "ThreateningEventsNear",
  "channelExecutionEpochTime": 1593142020436,
  "results": [
    {
      "result": {
        "event_info": {
          "eid": uuid("82e61d25-4cad-0632-3d8d-148e71cb50bf"),
          "name": "OC Marathon",
          "location":
             point("33.66100302712824, -117.83950620703125"),
          "event_duration": duration("PT10S"),
          "radius_km": 3.57746886883645
        },
        "tweet_distance_km": 4.854786471222485,
        "event_distance_km": 5.683937048494775,
        "oid": 0,
        "tweet_content": {
          "text": "Saul Goodman builds SKS, and Todd Alquist fires
                AK47, but Skyler White sells Cabbage.",
          "area_name": "UCI",
          "location": point("33.64921228736088,-117.84181977473024"),
          "threatening_rating": 2,
          "user_registered_weapon": [ "AR10" , "AK47", "GLOCK21" ]
        }
      },
      "channelExecutionTime": datetime("2020-06-26T03:27:00.436Z"),
      "subscriptionId": uuid("82e61d25-47ad-0632-3e5c-22b3cb7d7df4"),
      "deliveryTime": datetime("2020-06-26T03:27:00.437Z")
    }
  ]
}
```

Fig. 30. The generated threatening event notification from OCSD

In the UCI BAD AlertsOnCampus channel, threatening tweets are combined with on-campus building information and security station information to produce alerts. The threatening tweet in Figure 28 is near the building "Student Center", so UCI BAD produces a notification to alert people around this building as shown in Figure 31 (labeled 5 in Figure 26). The building information is attached to the notification. There are two security stations nearby, so the system attaches their information with their distances, ordered by their distances to the threatening tweet. Everyone subscribing to the AlertsOnCampus channel will receive this notification.

VII. BAD Islands Tour and Evaluation

To illustrate how BAD applications can be built with BAD islands and to visualize the process of data flowing through multiple systems and becoming notifications for subscribers, we have created three dashboards for each organization based on our prototype, as shown in Figure 32.

```
{
  "dataverseName": "uci",
  "channelName": "AlertsOnCampus",
  "channelExecutionEpochTime": 1593142024344,
  "results": [
    {
      "result": {
        "buildingInfo": {
          "bid": uuid("82e61d25-43ad-0632-45d0-0ba5366832d9"),
          "name": "Student Center",
          "area": rectangle("33.64811430275051, -117.84332027249145
                    33.649382536086605,-117.84153928570557")
        },
        "stationDist": [
          {
            "stationInfo": {
              "sid": 1,
              "location":
                 point("33.64792551859947, -117.84013290702327"),
              "name": "Station # 1"
            },
            "dist_km": 0.21216259109805177
          },
          {
            "stationInfo": {
              "sid": 0,
              "location":
                 point("33.646866723393266, -117.84170161534618"),
              "name": "Station # 0"
            },
            "dist_km": 0.23485382616041114
          }
        ],
        "tweetContent": {
          "text": "Saul Goodman builds SKS, and Todd Alquist fires
                    AK47, but Skyler White sells Cabbage.",
          "area_name": "UCI",
          "location": point("33.64921228736088,-117.84181977473024"),
          "threatening_rating": 2,
          "user_registered_weapon": [ "AR10" , "AK47", "GLOCK21" ]
        }
      },
      "channelExecutionTime": datetime("2020-06-26T03:27:04.344Z"),
      "subscriptionId": uuid("82e61d25-0ead-0632-4717-e17b6a912fa6"),
      "deliveryTime": datetime("2020-06-26T03:27:04.345Z")
    }
  ]
}
```

Fig. 31. The generated on-campus alert from UCI

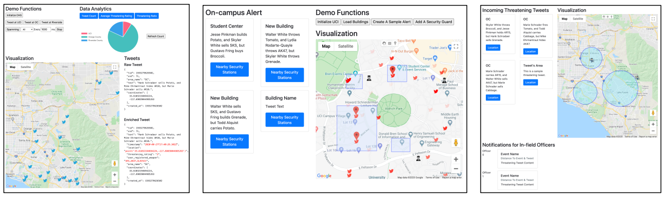

Fig. 32. An overview of BAD islands dashboards

Due to space limits, instead of describing the features on each dashboard in detail, we will focus on the Visualization Panel of the OCSD Dashboard, shown in Figure 33, to illustrate how threatening tweets go from DHS BAD to OCSD BAD and how OCSD BAD combines threatening tweets with other local information for its subscribers.

The Visualization Panel contains a map that shows the incoming threatening tweets, local events, produced threatening events, and in-field officers' movements. The map contains a control bar at the top (highlighted in a blue box) so dashboard users can navigate the map, add a new event, and add a new in-field officer. A new event can be added by drawing a circle on the map indicating the event's area. An officer can be added by dropping an officer icon on a preferred location on the map. Information about the created events and officers is updated in the underlying OCSD BAD system accordingly. An added officer moves around the map randomly and continuously sends its current location to the OCSD BAD system. One can change an officer's location by dragging the officer's icon to a new place on the map.

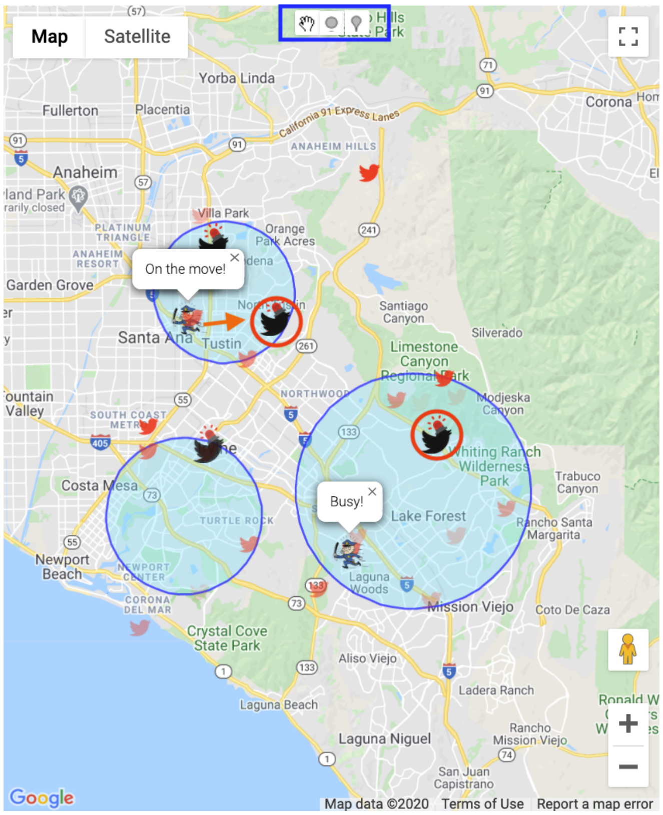

Fig. 33. Visualization panel of OCSD dashboard

We use a red tweet icon to mark the threatening tweets received from DHS BAD and a black tweet icon for the threatening events detected by OCSD BAD. When an in-field officer receives a threatening event notification (as highlighted in the red circles in the figure), the officer randomly decides whether to go to the threatening event's location for further investigation or to stay at his or her current location. The officer's decision pops up as a small information window, as shown in the figure. If the officer decides to go, he or she moves gradually towards the tweet's location, as the upper officer does in the figure.

In addition to the dashboards, we also conducted a simple experiment to measure the tweet propagation delays in our prototype system, starting from the posting of new tweets to the receipt of the localized notifications by subscribers on each island. We deployed the prototype on a three-node cluster, one node per island, where each node had a Dual-Core AMD Opteron Processor 2212 2.0 GHz, 8 GB of RAM, and a 900 GB hard disk drive. We used the statements described in Section VI to configure the nodes.

The information propagation times for BAD islands depend on the complexity of the computations in the pipeline (data enrichment and channel computation) and on the specified channel period durations. In our experiments, we used the same channel period for all three channels, testing two different channel periods (1s and 2s). Since channels execute once per each channel period, for each channel execution, we measured the average delay for threatening tweets delivered to subscribers in this channel execution. We let all channels complete 50 executions and kept track of the average delays throughout the process. On DHS BAD, tweets were set to arrive at 10 tweets per second, and half of the tweets contained at least one threatening word. On OCSD BAD, every threatening tweet had an event nearby. OCSD had 100 in-field officers constantly updating their locations and subscribing to nearby threatening events. On UCI BAD, every threatening tweet was close to an on-campus building. UCI had 5 on-campus security stations and 100 subscriptions subscribing to on-campus alerts. The delays are shown in Figure 34.

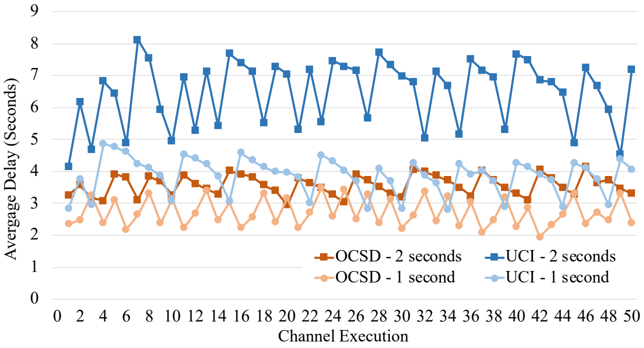

Fig. 34. Delays of threatening tweets for OCSD and UCI BAD subscribers

Clearly, the subscribers at OCSD and UCI are able to receive localized threatening tweets of interest in a timely manner, while the delays are relatively stable, especially for the 1s channel period. When the channel period is increased to 2s, the delays increase since the system batches more incoming tweets per channel execution; while this would increase the delay for subscribers, it would also increase the system scalability under higher loads. UCI BAD in general has higher delays than OCSD BAD due to a more complex computation and more local information added to local threatening tweets.

VIII. Conclusions

In this work, we have focused on enabling users to declaratively create scalable data sharing services between different BAD systems. We looked at an example use case in which two local organizations (OCSD and UCI) would like to get data from a third organization (DHS) in order to provide BAD services to their subscribers. We discussed several possible ways of supporting this use case and proposed using data feeds and data channels for bridging BAD systems. We extended the BAD system with *BAD brokers* to simplify data exchanges between channels and feeds and *BAD feeds* to help users create bridges between different BAD systems. We detailed a three-island prototype to show how BAD islands can be bridged together. We demonstrated how users can easily build such systems with declarative statements, and we used an example to show how data and events flow within the system. We built a set of dashboards based on our prototype to concretely illustrate how BAD islands share data and support BAD applications with localized information, and we conducted an experiment to examine the delays in the prototype system.

Acknowledgment

This research was partially supported by NSF grants IIS-1447826, IIS-1447720, IIS-1838222, IIS-1838248, CNS-1924694 and CNS-1925610.

References

[1] R. Bryant, R. H. Katz, and E. D. Lazowska, "'Big-Data Computing': Creating revolutionary breakthroughs in commerce, science and society," 2008.

[2] K. Shvachko, H. Kuang, S. Radia *et al.*, "The Hadoop distributed file system," in *IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST 2012, Lake Tahoe, Nevada, USA, May 3-7, 2010*, 2010, pp. 1–10.

[3] C. Olston, B. Reed, U. Srivastava *et al.*, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2008, pp. 1099–1110.

[4] A. Thusoo, J. S. Sarma, N. Jain *et al.*, "Hive - A warehousing solution over a map-reduce framework," *PVLDB*, vol. 2, no. 2, pp. 1626–1629, 2009.

[5] M. Zaharia, M. Chowdhury, T. Das *et al.*, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation*, 2012, pp. 15–28.

[6] P. T. Eugster, P. Felber, R. Guerraoui *et al.*, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, 2003.

[7] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: fault-tolerant streaming computation at scale," in *ACM SIGOPS 24th Symposium on Operating Systems Principles*, M. Kaminsky and M. Dahlin, Eds. ACM, 2013, pp. 423–438.

[8] P. Carbone, A. Katsifodimos, S. Ewen *et al.*, "Apache Flink™: Stream and batch processing in a single engine," *IEEE Data Eng. Bull.*, vol. 38, no. 4, pp. 28–38, 2015.

[9] M. J. Carey, S. Jacobs, and V. J. Tsotras, "Breaking BAD: a data serving vision for big active data," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, 2016, pp. 181–186.

[10] S. Jacobs, X. Wang, M. J. Carey, V. J. Tsotras, and M. Y. S. Uddin, "BAD to the bone: Big active data at its core," *VLDB J.*, 2020.

[11] S. Jacobs, M. Y. S. Uddin, M. J. Carey *et al.*, "A BAD demonstration: Towards big active data," *PVLDB*, vol. 10, no. 12, pp. 1941–1944, 2017.

[12] X. Wang, "Activating Big Data at Scale," Ph.D. dissertation, University of California, Irvine, USA, 2020.

[13] X. Wang, M. J. Carey, and V. J. Tsotras, "Subscribing to Big Data at scale," *arXiv preprint arXiv:2009.04611*, 2020.

[14] S. Wolfert, "Study on data sharing between companies in europe," 2018, [Online; accessed Jul-12th-2020].

[15] W. K. Michener, S. Allard, A. E. Budden *et al.*, "Participatory design of DataONE - enabling cyberinfrastructure for the biological and environmental sciences," *Ecol. Informatics*, vol. 11, pp. 5–15, 2012.

[16] R. Rice, "DISC-UK datashare project," in *Technology of Data: Collection, Communication, Access and Preservation*. IASSIST, 2008.

[17] E. Scaria, A. Berghmans, M. Pont, C. Arnaut, and S. Leconte, "Study on data sharing between companies in Europe," *A study prepared for the European Commission Directorate-General for Communications Networks, Content and Technology by everis Benelux*, vol. 24, 2018.

[18] S. Alsubaiee, Y. Altowim, H. Altwaijry *et al.*, "AsterixDB: A scalable, open source BDMS," *PVLDB*, vol. 7, no. 14, pp. 1905–1916, 2014.

[19] D. Chamberlin, *SQL++ For SQL Users: A Tutorial*. Couchbase, Inc., 2018, (Available at Amazon.com).

[20] D. B. Terry, D. Goldberg, D. A. Nichols *et al.*, "Continuous queries over append-only databases," in *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, 1992, pp. 321–330.

[21] W. Y. Alkowaileet, S. Alsubaiee, M. J. Carey *et al.*, "End-to-end machine learning with Apache AsterixDB," in *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, 2018, pp. 6:1–6:10.

[22] X. Wang and M. J. Carey, "An IDEA: an ingestion framework for data enrichment in AsterixDB," *PVLDB*, vol. 12, no. 11, pp. 1485–1498, 2019.

[23] R. Grover and M. J. Carey, "Data ingestion in AsterixDB," in *EDBT Conf.*, 2015.

[24] W. Y. Alkowaileet, S. Alsubaiee, M. J. Carey *et al.*, "Enhancing Big Data with semantics: The AsterixDB approach (poster)," in *12th IEEE International Conference on Semantic Computing, ICSC*. IEEE Computer Society, 2018, pp. 314–315.