snews

Skylar(Yiyang) Xu

Jul 20, 2021

CONTENTS

CHAPTER

ONE

USER'S GUIDE

1.1 Installation

You can install snews via pip or from source.

To install with pip:

```
pip install -U snews
```

To install from source:

```
tar -xzf snews-x.y.z.tar.gz
cd snews-x.y.z
python setup.py install
```

1.2 Quickstart

1.2.1 Run SNEWS 2.0

After installing the module, the command that does the magic is:

```
snews model
```

The two required options are:

- -f: the .env file for required environment variables.
- –no-auth: True to use the default .toml file. Otherwise, use Hopskotch authentication in the .env file.

So an example command would be

```
snews model --env-file config.env --no-auth
```

Configuration

The user should create a .env file and pass the file path to the -f option when running SNEWS 2.0. The .env file should include the following:

The definition of these environmental variables are:

- COINCIDENCE_THRESHOLD: maximum time (s) between messages for them to be considered coincident
- MSG_EXPIRATION: maximum time (s) that a message will be stored in the database cache before expiring
- TIME_STRING_FORMAT: the string format of time in all SNEWS messages.
- DATABASE_SERVER: the database server to that SNEWS 2.0 connects to in order to store messages for processing. In the current version, the app takes in a **MongoDB** server.
- NEW_DATABASE: "True" to drop all previous messages and "False" to keep them.
- OBSERVATION_TOPIC: the Hopskotch topic for detectors to publish messages to.
- TESTING_TOPIC: the optional topic for testing.
- ALERT_TOPIC: the Hopskotch topic for SNEWS 2.0 to publish alert messages to the detectors.

Access to Hopskotch

To configure a .toml file for hop-client module, follow the steps documented at https://github.com/scimma/hop-client and specify –default-authentiation as False.

Otherwise, in the .env file, include the following:

```
USERNAME=username
PASSWORD=password
```

where "username" and "password" are user credentials to Hopsckoth.

Generate Messages

`snews generate` can be used to simulate real-time messages from experiments:

```
snews generate
```

with options

- –env-file: the .env file for configuration.
- –rate: the rate of messages sent in seconds (e.g. 2 means one message every 2 seconds).
- –alert-probability: the discrete probability that the message is significant.
- –persist: continually send messages. If not specified, send only one message.

For example, to continuously publish two messages per second, each with a 10% probability of being a significant, enter:

```
snews generate --env-file config.env --rate 0.5 --alert-probability 0.1
```

Alternative Instances

If the user does not have access to the Hopskotch or MongoDB server or both, running local instances is a alternative choice.

- To run a Kafka instance, run the following in the shell

```
docker run -p 9092:9092 -it --rm --hostname localhost scimma/server:latest --noSecurity
```

and pass the following Kafka server to SNEWS 2.0

```
kafka://dev.hop.scimma.org:9092/USER-TOPIC
```

- To run a MongoDB instance, either run

```
docker run -p 27017:27017 -it --rm --hostname localhost mongo:latest
```

or run

```
pip install -U mongoengine
```

and pass the following MongoDB server to SNEWS 2.0

```
mongodb://localhost:27017/
```

1.3 Architecture

The SNEWS 2.0 implementation has three major components:

1.3.1 Model

The model initializes a Decider object and opens up a kafka stream through hop-client to read in messages from detectors. The model would evoke different processing functions depending on the message type. The message types and corresponding processing algorithms are stored as a mapping in

```
self.mapping = {
        SNEWSObservation.__name__: self.processObservationMessage,
        SNEWSHeartbeat.__name__: self.processHeartbeatMessage
}
```

Upon receiving an observation message, it stores the message by calling methods of the decider and then runs the coincidence requirement check through the decider's methods. If the deciding function indicates the possibility of a

potential supernova, the model generates an alert message and sends it to all detectors through a differet Hopskotch stream.

Each detector are required to periodically send heartbeat messages. The model keeps a record of which detectors are on or off and removes detectors from which it has not heard back in a long time. When receiving a heartbeat message, the model updates the status of status and machine time of the detector included in this message.

1.3.2 Decider

A decider consists of a Database Storage object and an implementation of the SNEWS coincidence requirement protocol (the `deciding()` function).

Pseudocode for the deciding protocol logic is:

The API of the decider class is defined as

```
class IDecider(object):
def deciding(self):
    pass

def addMessage(self, time, neutrino_time, message):
    pass

def getAllMessages(self):
    pass
```

1.3.3 Database Storage

This is an object for storing and queueing observation messages received by the model. In order to support the coincidence requirement check and future SNEWS usage, the storage object should have the functionality to received timestamped messages. The API of this class is

```
class IStorage(object):
def insert(self, time, neutrino_time, message):
    pass

def getAllMessages(self):
    pass

def cacheEmpty(self):
    pass

def getMsgFromStrID(self, post_id):
    pass
```

For the first release/pre-release of SNEWS 2.0, MongoDB is used to implement this IStorage interface. TTL indexes are used to expire messages. Two MongoDB collections are created here, with one storing all messages and the other one acting as a timed cache for coincidence requirement check.

1.4 Messages

1.4.1 SNEWS Custom Messages

SNEWS 2.0 makes use of the fact that the hop-client module supports adding custom message formats as plugins. SNEWS 2.0 implements a custom message plugin: https://github.com/SNEWS2/hop-plugin-snews The message format itself is described in the SNEWS plugin documentation: https://hop-plugin-snews.readthedocs.io/en/latest/user/messages.html

For documentation on the hop-client custom message plugins, see https://hop-client.readthedocs.io/en/latest/user/models.html

1.5 SNEWS Protocols

1.5.1 Coincidence Requirement

Upon observing a potential pre-supernova phenomena, the detector generates an observation message and publish it. The SNEWS 2.0 server reads in the message and run the coincidence requirement check defined by SNEWS astronomers. The first version of the protocol compares the time and locations of observations among unexpired messages. The simplified version of the algorithm is

```
If there're multiple messages within the last 24 hours:
    Then iterates through messages to check if any two or more are within 10s
        If yes:
            verify locations different (as long as at least two are in different␣
↪locations)
                If the locations are different:
                    return true
                Otherwise:
                    return false
            If not, no-op or print a message
        if no:
            no-op
```

In future releases, the coincidence requirement protocol will likely include triangulations to identify coordinates of the possible supernova.

CHAPTER

TWO

API REFERENCE

2.1 snews API

2.1.1 snews.storage

class snews.storage.**IStorage**

class snews.storage.**MongoStorage**(*msg_expiration*, *datetime_format*, *server*, *drop_db*)

getAllMessages()
sort by pymongo.ASCENDING (1) gives dates from old to recent sort by pymongo.DESCENDING (-1) gives dates from recent to old :return:

insert(*sent_time*, *neutrino_time*, *message*)
Need to CONVERT STRING TIME to DATETIME OBJECT :param time: :param message: MUST be a dictionary :return:

2.1.2 snews.decider

class snews.decider.**Decider**(*coinc_threshold=10*, *msg_expiration=120*, *datetime_format='%y/%m/%d %H:%M:%S'*, *mongo_server='mongodb://localhost:27017/'*, *drop_db=True*)

addMessage(*message*)

Parameters **message** – message from a hopskotch kafka stream

Returns

deciding()
Implements the SNEWS coincidence requirement protocol to check cached messages and determine whether an alert message will be sent.

Returns True or false indicating a coincidence between messages

getAllMessages()
Get all messages in history. :return:

getCacheMessages()
Get messages that have not expired. :return:

class snews.decider.**IDecider**

2.1.3 snews.model

snews.model.**main**(*args*)
main function

snews.model.**validateJson**(*jsonData*, *jsonSchema*)
Function for validate a json data using a json schema. :param jsonData: the data to validate. :param jsonSchema: the schema assumed to be correct. :return: true or false

2.1.4 snews.generate

class snews.generate.**Detector**(*detector_id*, *location*)

property detector_id
Alias for field number 0

property location
Alias for field number 1

snews.generate.**generate_message**(*time_string_format*, *detectors*, *alert_probability=0.1*)
Generate fake SNEWS alerts/heartbeats.

snews.generate.**main**(*args*)
generate synthetic observation/heartbeat messages

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX