The AMOS database system

Storing data, one meteor at a time

Mgr. Martin Baláž

DAPEM FMPH UK

2019–04–17

Overview

Contents

objective	What are we doing?
motivation	Why?
implementation	How?
results	What have we done?

Objective

- to build a database system for storing real meteor data
- it should be

web-based	accessible from anywhere
comprehensible	we are able to identify interesting events easily
autonomous	integrate the entire processing pipeline

Motivation

Current state is a disaster

Motivation

Current state is ~~a disaster~~ suboptimal

- no unified file format
- data spread across multiple computers
- analysis next to impossible
    - and I badly missed it in diploma thesis

Data

Acquisition

Data are acquired by AMOS cameras

- **UFOCapture** by SonotaCo
- **AMOS** by Kvant
    - will be ready in about two months...

Acquisition

Data are acquired by AMOS cameras

- **UFOCapture** by SonotaCo
- **AMOS** by Kvant
  - will be ready in about two months...
  - ...for the last three years

Pre-processing

Pre

- meteor recognition and data extraction
  - position in the sky
  - magnitude
  - angular speed
  -

Retrieval

Currently:

- UFOCapture launches a `bat` file
- mail transfer via SMTP (e-mail)
- processed by `charon`

Proposal:

- video capturing software forwards data to a daemon
- sent over HTTP in a POST request

Retrieval

Currently:

- UFOCapture launches a `bat` file
- mail transfer via SMTP (e-mail)
- processed by `charon`

Proposal:

- video capturing software forwards data to a daemon
- sent over HTTP in a POST request
- we need an **API**
  - station submits a new `Sighting`
  - website retrieves a list of `Meteor` s
  - daemon periodically computes `Meteor` data
  - other internal use (maps, analyses...)

Processing

Data need to be **validated** and **processed**

- remove invalid data (false detections, ...)
- compute `Meteor`s from multiple `Sighting`s
-

Storage

At this point data are ready to be stored

- in a **structured** and **semantic** way
- consistency is **enforced** at all times
- auxiliary data
  - housekeeping
  - statistics

Data retention

To prevent problems down the line, we should

- keep **all** raw data available
- never delete anything (unless provably incorrect)
- enable raw data re-processing
  - for example to import data from an offline station

Housekeeping

Def: Data of **low scientific value** but **high operational importance**

- environment
  - temperature
  - humidity
  - pressure
  - ...

Housekeeping

Def: Data of **low scientific value** but **high operational importance**

- environment
  - temperature
  - humidity
  - pressure
  - ...
- network
  - system uptime
  - network connection
  - UPS status
  - ...

- displayed on dashboard

Database

The database

Problem: We need a way to visualize and comprehend the data

The database

Problem: We need a way to visualize and comprehend the data **Solution**: A database and a simple web interface

- provides a basic framework for data operations
- much more user-friendly than a bare directory listing
- much easier to retrieve, sort and analyze the data

Design

Underlying data are well-structured and suitable for an **object-relational** database

- we require

text files PostgreSQL SQLite MySQL

Design

Underlying data are well-structured and suitable for an **object-relational** database

- we require
  - full ACID compliance

text files PostgreSQL SQLite MySQL

Design

Underlying data are well-structured and suitable for an **object-relational** database

- we require
  - full ACID compliance
  - free

PostgreSQL SQLite MySQL

Design

Underlying data are well-structured and suitable for an **object-relational** database

- we require
  - full ACID compliance
  - free
  - able to scale well

PostgreSQL MySQL

Design

Underlying data are well-structured and suitable for an **object-relational** database

- we require
  - full ACID compliance
  - free
  - able to scale well
  - decoupled from the rest of the system

PostgreSQL

Design

Underlying data are well-structured and suitable for an **object-relational** database

- we require
  - full ACID compliance
  - free
  - able to scale well
  - decoupled from the rest of the system
  - I like it

PostgreSQL

ORDBMS

- data are stored in **relations** (tables)
- each column stores the same **property**
- each row stores a single **entity** (object)
- each object has an identifier (primary key)
- fields may point to other tables (foreign keys)
- data are accessed and manipulated using a query language

ORDBMS

```
SELECT ”id”, ”timestamp”, ”magnitude” \
FROM ”meteors” \
WHERE ”timestamp” BETWEEN ”2019-04-16 15:00:00” AND ”2019-04-17 09:00:00” \
ORDER BY ”magnitude” ASC LIMIT 5;
```

ORDBMS

```
SELECT ”id”, ”timestamp”, ”magnitude” \
FROM ”meteors” \
WHERE ”timestamp” BETWEEN ”2019-04-16 15:00:00” AND ”2019-04-17 09:00:00” \
ORDER BY ”magnitude” ASC LIMIT 5;
```

```
356, ”2019-04-17 03:45:10”, -5.8
728, ”2019-04-17 04:14:23”, -3.2
456, ”2019-04-16 23:56:04”, -2.7
908, ”2019-04-17 01:23:45”, -2.5
854, ”2019-04-16 21:58:35”, -2.2
```

Models

We should be able to naturally translate the real world to models

- `Meteor`
- `Sighting`
- `Station`
- `Subnetwork`
- `Country`

Model `Sighting`

Describes the **sighting** of a single meteor by an AMOS camera

- identifier
- observed projected position on the sky
- three coordinates
  - azimuth
  - altitude
- at three moments
  - beginning
  - maximum brightness
  - end
- maximum apparent magnitude

Model `Sighting` – extras

- miscellaneous computed information
  - arc length
  - duration
  - Sun and Moon info
    - position
    - elongation
    - magnitude
- visualisation
  - real photograph from AMOS
  - corresponding simulation (?)

(link)

Model `Meteor`

Describes the actual **event**, an atmospheric entry of a meteoroid particle

- **timestamp**
- true geographic location
  - again, four coordinates
    - **timestamp**
    - **latitude**
    - **longitude**
    - **altitude**

Model `Meteor`

Describes the actual **event**, an atmospheric entry of a meteoroid particle

- **timestamp**
- true geographic location
  - again, four coordinates
    - **timestamp**
    - **latitude**
    - **longitude**
    - **altitude**
  - at three moments
    - beginning
    - maximum brightness
    - end

Model `Meteor` – auxiliary data

- true **trail length**
  - this is not well-defined
- computed maximal **absolute magnitude** (least-squares)
- **visualisation**
  - KML file for Google Earth
  - online map (OpenLayers)

(link)

Visualisation

Overview

We have implemented a website in Python/Django

- webserver + CRUD operations
- administration interface (courtesy of Django)
- REST framework

System in operation

- Dashboard
- Meteor list
- Sighting list
- Example of a sighting
- Example of a meteor
- Example of a meteor path

admin!

Thank you for your attention

Above all else, show the data.

Edward R. Tufte
The Visual Display of Quantitative Information, 1983

References

- **Jones, W.; Halliday, I.**: Effects of Excitation and Ionization in Meteor Trains. MNRAS 320, 4, 417–423 (2001)