docs.audioblast.org

Ed Baker

2024-01-12

Contents

Chapter 1

About audioBlast

audioBlast is a project to make the world's collections of bioacoustic and ecoacoustic recordings findable, accessible, and searchable.

Chapter 2

Introduction

2.1 Design Philosophy

The technical design of audioBlast pushes computation down the stack as far as is conveniently possible. Ideally this will often be the database. This results in a reduction in network traffic, which can be significant if returning a large number of analyses.

Some core parts of the audioBlast system (i.e. the ingest process and analyse process) can communicate directly with the database, and all insertion or updating of records must be performed through one of these processes.

In general, and for all end-user cases, reading of data should be performed via the API (https://api.audioblast.org). The API is a lightweight database wrapper that automatically handles pagination of results and caching of slow queries. Various utilities provide access to the API in different environments, including the sonicscrewdriver package for the R environment. The API is also used to provide the data search and discovery tools at https://audioblast.org.

Table 2.1: Simplified illustration of audioBlast abstraction layers from bottom to top.

	Ingest	Analyse
MySQL R API Users	Triggers INSERT UPDATE r-dbi abdb	Stored Procedures INSERT UP r-dbi abdb

2.2 Analyse everything once, in advance

Many bioacoustic and ecoacoustic studies make use of a standard set of analyses. These are computed in advance by audioBlast, allowing the analyses to be instantly available and searchable. An additional advantage is that the computational resource needed to compute analyses need only be performed once.

Chapter 3

audioBlastIngest

The ingest service is responsible for bringing data about recordings, deployments, annotation, taxa and traits into the audioBlast system. Data sources are specified as modules within the audioBlast API, which give details of each data source and the processing that is expected to be performed by the ingest process. The ingest process itself is managed on audioBlast infrastructure by the audioBlastIngest package. This package obtains data from the sources specified in the API modules, performs and requested transformations, and loads the data into the audioBlast database.

Note: The database generates analysis tasks using triggers as data is inserted or updated, however the analyses are performed by a separate processes to the ingest.

Updates are handled periodically (generally nightly) by the ingest server. Creating an ingest task is a simple three step process.

1. Third party publishes data online.

2. audioBlast team work with third party to create an API module to map the published data on to audioBlast data structures.

3. The audioBlastIngest package handles the import of the data.

3.1 Source modules

3.1.1 Fields

3.1.2 Mapping columns

3.1.2.1 Optional processing done by audioBlast

3.1.2.1.1 Specifying processors

3.1.2.1.1.1 SourceR: Automatically prepend a source column

3.1.2.1.1.2 date2dateAnTime(): Date and time mapping You can map datetime fields to Date and use the date2dateAndTime() processor to split them into separate date and time fields.

Chapter 4

audioBlastAnalyse

audioBlastAnalyse is an R package that handles the analysis of recordings. While some analyses (e.g. annotations) can be imported during the ingest process this is not true for all sources of recordings.

4.1 Requirements

The analysis system requires a computer running the R environment for statistical computing. The package will import all dependencies on installation.

The package is tested on recent MacOS, Linux, and Windows systems.

If the analysis system is to write directly to the audioBlast database then it must additionally be on the NHM network (including via the VPN).

4.2 Development

The package is hosted on the audioBlast GitHub at https://github.com/audioblast/audioBlastAnalyse.

4.3 Installation

Installation of the package requires the devtools package.

```
install.packages("devtools")
```

4.3.1 From GitHub

```
devtools::install_github("audioblast/audioBlastAnalyse")
```

4.3.2 From R working directory

```
devtools::install()
```

Chapter 5

Adding new analyses

The process for adding new analyses is:

1. Create a new database table for storing the analysis, following the column tyes and names of 'analysis-aci'.
2. Add the analysis table to the deleteAllAnalyses() function of audioBlast-Analyse R package.
3. Implement the analysis in audioBlastAnalyse R package.
4. Implement a module for https://api.audioblast.org to provide API access to the analysis.
5. Make arrangements for existing recordings to be queued for analysis if required.

Chapter 6

The Task Queue System

The Task Queue System maintains a list of outstanding analysis tasks in the database and is responsible for assigning tasks to analysis processes.

The database implementation is designed so that it can be utilised by either audioBlastAnalyse or in the future by an analysis package in another language (e.g. a Python analysis suite).

6.1 Database Implementation

The system is implemented as two tables in the database, `tasks` and `tasks-progress`. Tasks are assigned using the stored procedure `get-tasks()` for local files, or `get-tasks-by-file()` for web files. The `get-tasks()` procedure will assign `n` random tasks to the analysis process, whereas `get-tasks-by-file()` will return all outstanding tasks for a given file. The latter procedure removes the need for repeated downloads of the same file.

6.1.1 `tasks` table

The table `tasks` is a list of outstanding analysis tasks. It is populated automatically by triggers on the `recordings` and `recordings-calcuated` table. Analysis tasks are grouped into sections (recordings_calculated, sounscapes_minute and soundscpaes_second). The `recordings-calculaed` table has a TINYINT column for each of these, that is set to `1` once the task is completed.

6.1.2 `tasks-progress` table

The table `tasks-progress` is populated by the stored procedure `get-tasks()` when an analysis process requests outstanding tasks. It is used to prevent the same task being analysed to multiple analysis processes. Tasks from an unresponsive process (e.g. a process which has crashed) can be reassigned after an hour of inactivity by that process.

6.1.3 `get-tasks()` stored procedure

The stored procedure `get-tasks(process_id, n, source)` is used to assign `n` tasks to the analysis process `process_id`. Initially a call to `_quickMaintain()` is performed to deallocate time-expired tasks. It inserts `n` random unassigned tasks from the `tasks` table into the `tasks-progress` table along with the process_id and the current timestamp. It returns via a SELECT statement these tasks.

6.1.4 `get-tasks-by-file()` stored procedure

Initially a call to `_quickMaintain()` is performed to deallocate time-expired tasks. The stored procedure `get-tasks(process_id, source)` then assigns all outstanding tasks for a randomly chosen file to the analysis process `process_id`. It inserts these tasks from the `tasks` table into the `tasks-progress` table along with the process_id and the current timestamp. It returns via a SELECT statement these tasks.

6.1.5 `delete-task()` stored proecure

The stored procedure `delete-task(process_id, source, id, task)` marks a task as complete by deleting it from the `tasks` table. A trigger in the `tasks` table will delete the matching row from `tasks-progress`. The started time of any remaining tasks assigned to the same `process_id` in the `tasks-progess` table will be updated to the current time. A call is made to the stored procedure `_quickMaintain()` which will perform routine maintenance tasks, including removing any expired tasks in the `tasks-progress` table.

6.2 audioBlastAnalyse Implementation

The analysis suite audioBlastAnalyse has several functions that implement this task queue in the R language.

All these functions require a database connector (`db`) as a parameter, and a unique `process_id` for the analysis process. In the majority of cases these functions will be called automatically from within a call to the main `analyse()` function, where the database connector will already be configured and a process identifier is automatically generated.

6.2.1 Legacy mode

On some older Linux operating systems the RMariaDB package and libraries it builds against have issues dealing with stored procedures. For the time being a legacy mode is provided that implements the stored procedures as a sequence of queries. Setting `legacy=TRUE` as a parameter to either of the fetch queries will activate this mode.

6.2.2 Getting tasks

The `get-tasks()` stored procedure is accessed by using `fetchDownloadableRecordings()`, and `get-tasks-by-file()` by using `fetchDownloadableRecordings()`.

```
fetchDownloadableRecordings(db, source, process_id, legacy=FALSE)
fetchUnanalysedRecordings(db, source, process_id, legacy=FALSE)
```

6.2.3 Removing completed tasks

```
deleteToDo(db, source, id, task, process_id)
```

Chapter 7

Analysis infrastructure

7.1 Natural History Museum High-Performance Compute

The Natural History Museum has a high-performance compute cluster that can be used for audio analysis. The cluster is managed using SLURM and environments are managed using conda.

7.1.1 conda environments

7.1.1.1 ab-r2

This environment contains R packages for acoustic analysis, and also has audioBlastAnalyse installed to manage analyses.

```
conda activate ab-r2
```

7.1.1.2 birdnet

This is a Python environment for running BirdNET analyses.

```
conda activate birdnet
```

Chapter 8

audioblast.org

The website provides a convenient search and browsing interface making use of api.audioblast.org. This server is hosted by the Natural History Museum, London.

Chapter 9

Search module API

The search module API allows for processing of a search term on audioblast.org, and handles the display of relevant search results. Modules are typically related to a particular data type (i.e. a table in the audioblast database), although there are more abstract modules that solely process search terms in order to allow them to be processed by other modules.

9.1 Search API class

9.1.1 init()

9.1.2 displayPrototype()

Create a region on the results page for displaying search results.

9.1.3 parse()

Parses the search term for relevant terms, and may modify terms.

9.1.4 display()

Handles the display of search results to the user.

9.1.5 searchSuggests()

Provides a short list of indicative search terms that may be handled by the module. This is used to suggest search terms on the audioblast.org homepage.

9.2 Current audioblast.org search modules

9.2.1 Pythia

Pythia uses an audioBlast API to match words in the search term to tokens using direct access to the audioBlast database.

9.2.2 Linnaeus: taxonomic name information and processing

The parse() function in Linnaeus takes a taxon name token identified by Pythia and converts it to a taxon name with rank token.

Linnaeus also handles the taxon info box, that provides search navigation through parent taxa.

9.2.3 King Solomon's Ring

King Solomon's Ring converts everyday terms relating to animal behavior into named trait with value tokens (e.g. silent to sound production method being equal to None.)

9.2.4 Rosetta

Rosetta handles the conversion of emoji in search terms to taxonomic names using the Phymoji package.

Chapter 10

Maintenence tasks

10.1 cron tasks

Every 5 minutes a cron job on the unp-audio-2 virtual machine updates the total analysis counts by calling the API endpoint /standalone/analysis/fetch_analysis_counts/ without caching (?cache=0).

Other slow-running queries that impact the speed of the user interface can be added here if necessary.

Chapter 11

Maintenence notices

These are created and displayed on the status.acousti.cloud server, at present by directly editing the table notices in the status MySQL database. The active column is set to 1 for active notices.