NoozyIA version 1.0 : Manuel de référence

Azim Roussanaly, Olfa Messaoud

Projet SmartVideo Grand Est

NoozyIA version 1.0 : Manuel de référence

Azim ROUSSANALY
Olfa MESSAOUD

Version de
23 février 2022

Table des matières

Chapitre 1

Architecture générale

Dans le cadre du projet SmartVideo Grand Est, qui consiste à créer la plateforme `noozy.tv` de diffusion de vidéos en ligne produits par des chaînes de la région Grand Est de la France, le LORIA a conçu et développé la composante **NoozyIA** qui comprend principalement un moteur de recommandations invoqué par la plateforme pour faciliter la navigation des utilisateurs et leur recommander un contenu pertinent.

Le moteur de recommandation se présente comme un service qui permet de fournir, aux développeurs de la plateforme, un ensemble d'algorithmes de recommandations nécessaires pour enrichir la plateforme noozy.tv avec des bandeaux de recommandations afin d'adapter, de personnaliser et de diversifier les contenus proposés aux utilisateurs (voir figure 1.1).

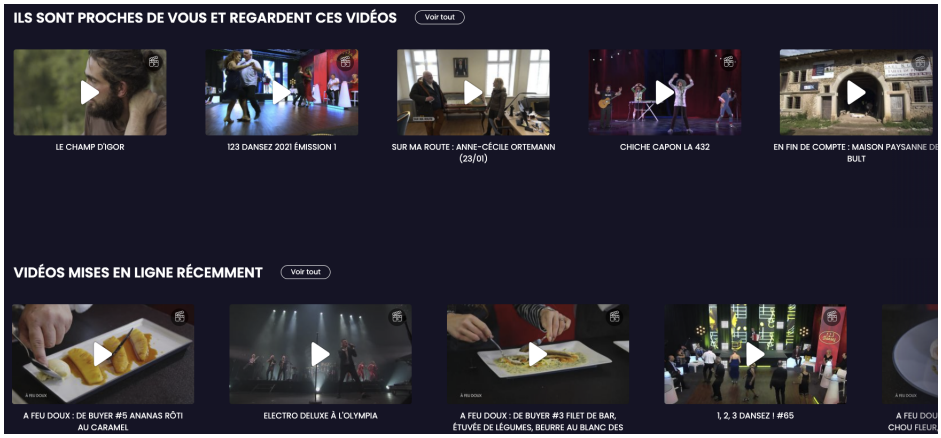

FIGURE 1.1 – Exemples de bandeaux sur noozy.tv

La figure 1.2 présente le flux des données échangées avec la composante NoozyIA.

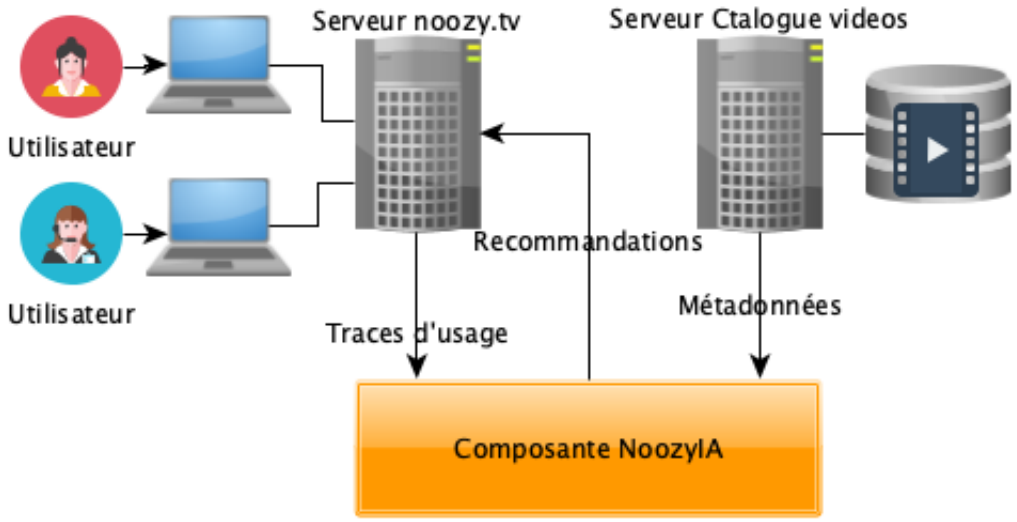

FIGURE 1.2 – Flux des données de NoozyIA

NoozyIA interagit avec la plateforme noozy.tv d'une part, en collectant les données d'usage par les utilisateurs de la plateforme et, d'autre part, en sollicitant des recommandations de vidéos afin de les présenter à ses utilisateurs. La collecte des traces d'usage est basée sur le standard **xAPI** [1].

NoozyIA reçoit également les méta-données décrivant les vidéos dans le catalogue des oeuvres disponibles pour la plateforme. Les standards utilisés pour l'échange sont, **OAI-PMH** pour le transport [6], et **Dublin Core** pour le format des méta-données [9], [8].

La composante NoozyIA est formée de plusieurs modules autonomes, le principe étant que chaque module pourrait être déployé sur une machine différente. Dans la version actuelle, ceux-ci sont principalement déployés parallèlement sur l'unique serveur *loria.kd-serveur.com*, mis à disposition par la société Kardham Digital, partenaire du projet SmartVideo Grand Est.

La figure 1.3 offre une vue schématisée de l'architecture générale de NoozyIA.

Les modules sont de deux types distincts :

- les modules qui se présentent comme des API RESTFul [10] : le Serveur xAPI et le Moteur de recommandations,
- les modules temporels qui se déclenchent à intervalles de temps régulier : le module Harvester (Moissonneur) et les modules Algorithmes

[]

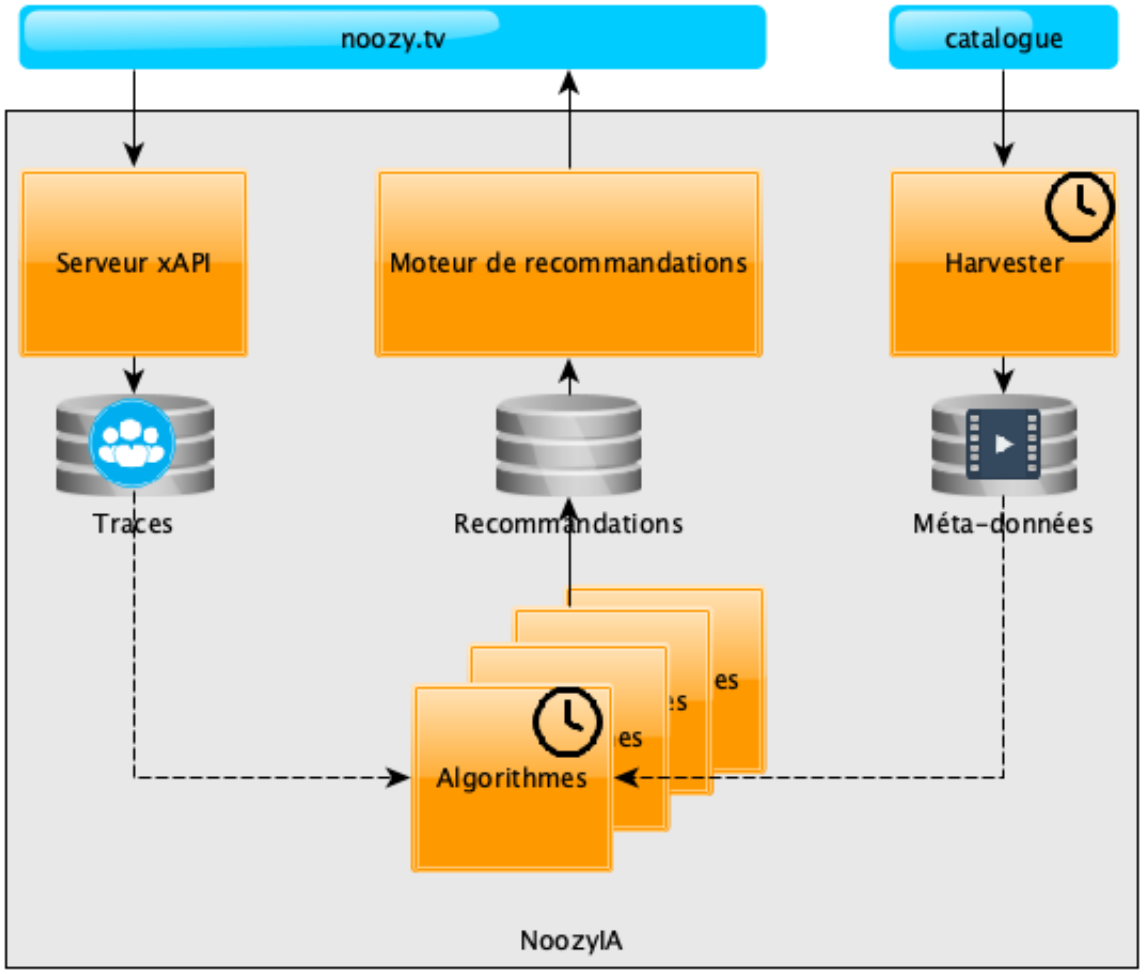

FIGURE 1.3 – Architecture de NoozyIA

1.1 Serveur xAPI

Un serveur xAPI est un serveur qui se conforme au standard xAPI. Ce serveur est destiné à être invoqué par la plateforme via des requêtes RESTFul dans le but de collecter, en temps réel, les traces correspondant à chaque action réalisée par un utilisateur. Une action, appelée aussi **verbe**, traduit, par exemple, une connexion à la plateforme, la recherche d'une vidéo ou la consultation d'une oeuvre. Une description technique de tous les verbes est disponible dans le document [5].

Il existe, sur marché, plusieurs serveurs xAPI. Certains sont fournis sous

licence Open Source, et d'autres sont des produits commerciaux. Dans le cadre du projet SmartVideo, nous avons fait le choix d'intégrer le produit **Open Source TRAX LRS V1** [7] dans la distribution de NoozyIA sachant qu'il pourrait être substitué par n'importe quel autre produit conforme au standard.

TRAX LRS repose sur une SGBD MySQL interfacée par une API REST à travers laquelle s'effectuent toutes les requêtes.

1.2 Harvester

Le moissonnage (Harvesting) consiste à collecter les méta-données de catalogues dans différents serveurs normalisés. Ici le standard retenu est **OAI-PMH**. Le partenaire Kardham Digital, en charge de la gestion du catalogue, a mis à notre disposition une API conforme à ce standard.

Remarque : Dans le cas de ce projet, il n'y a qu'une seule source de moissonnage.

Le Harvester est doté d'un *scheduler* (planificateur) qui est chargé de récupérer, avec une fréquence quotidienne, l'ensemble des méta-données des nouvelles vidéos enregistrées dans le catalogue partagé par l'ensemble des producteurs de la plateforme.

Les méta-données sont collectées au format standard **Dublin Core** s'appuient sur une hiérarchie de champs explicités, pour notre cas d'usage, dans le document [4].

De manière interne, le Harvester met régulièrement à jour une base de méta-données en s'appuyant sur une base REDIS.

1.3 Algorithmes

Il existe plusieurs modules **Algorithmes**, chacun faisant référence à un algorithme particulier de recommandations. Dans la version présente de NoozyIA, nous avons développé 10 modules différents :

1. **Most Recent :** retourne la liste des N dernières vidéos ajoutées à la base de données ou au catalogue des vidéos (les plus récents)
2. **Most Popular :** retourne la liste des N vidéos les plus regardées (les plus populaires) durant les 3 derniers mois
3. **Popular by Genre :** retourne la liste des N vidéos les plus regardées pour un genre donné pendant les 3 derniers mois

4. **Popular by Contributor :** retourne la liste des N vidéos les plus regardées pour une chaîne donnée durant les 3 derniers mois
5. **Popular by Thematic :** retourne la liste des N vidéos les plus regardées pour une thématique donnée durant les 3 derniers mois
6. **CF Nearby :** retourne la liste des N vidéos que les utilisateurs proches géographiquement de l'utilisateur courant ont regardé (sur la base de la localisation ou, à défaut, de son estimation à partir du numéro IP tracé)
7. **CF User Based :** retourne la liste des N vidéos appréciées des utilisateurs ayant un goût comparable à l'utilisateur courant (en utilisant l'algorithme K Nearest Neighbors)
8. **CF Item Based :** retourne la liste des N vidéos les plus fréquemment visionnées par les autres utilisateurs avec la dernière vidéo vue par l'utilisateur courant
9. **CB User Based :** considère les trois dernières vidéos visualisées par l'utilisateur courant pour calculer la similarité entre ces vidéos et les autres vidéos (en utilisant le contenu des champs *titre*, *mots-clefs*, *résumé*, *auteurs* ...) et recommander la liste des N vidéos les plus similaires
10. **CB Item Based :** retourne la liste des N vidéos les plus similaires à une vidéo donnée (par exemple celle qui vient d'être vue)
11. **Random :** sélectionne N vidéos au hasard (utilisé uniquement pour les tests)

Remarque 1 : Pour tous les algorithmes développés, les oeuvres recommandées appartiennent à l'ensemble des vidéos non encore visionnées par l'utilisateur donné en paramètre.

Remarque 2 : Une phase de post-traitement est appliquée pour diversifier la liste des contenus recommandés.

Remarque 3 : Chaque module calcule en avance les recommandations et les stocke dans une base REDIS. Le calcul est régénéré à intervalle régulier (dans la version actuelle, une fois par jour). Les recommandations pré-calculés serviront ensuite au module **Moteur de recommandations**.

Remarque 3 : Pour les algorithmes basés sur les items (Item Based : #8 et #10), les recommandations sont générées pour chaque vidéo du catalogue. Pour les autres algorithmes, le calcul est effectué pour chaque utilisateur de la plateforme.

1.4 Moteur de recommandations

Une fois que les algorithmes de recommandation sont exécutés et les résultats stockés dans une base REDIS, le **Moteur de recommandations** est en mesure de satisfaire les requêtes en provenance de la plateforme.

La base des recommandations pré-calculées est interfacée par une API REST dont les requêtes sont définies dans le document [3] qui décrit formellement les arguments associés à chaque algorithmes.

Voici un résumé de ces arguments :

- **n :** le nombre de vidéos à retourner, obligatoire pour tous les algorithmes
- **user_id :** identifiant de l'utilisateur concerné par la requête de recommandation (tous les algorithmes sauf #8 et #10)
- **video_id :** identifiant de la vidéo pour les algorithmes basés sur les items (algorithmes #8 et #10)
- **genre :** le genre, obligatoire pour l'algorithme Popular by Genre
- **contributor :** ou chaîne, obligatoire pour l'algorithme Popular by Contributor
- **thematic :** obligatoire pour l'algorithme Popular by Thematic

1.5 Tracking et Monitoring

A ces modules qui constituent le coeur de NoozyIA, nous avons rajouté un module de **Tracking et Monitoring** indépendant qui permet de suivre dans le temps la qualité des algorithmes déployés.

Pour ce faire, nous avons défini un indicateur de performance spécifique que nous appelons la **couverture**.

$$Coverage = \frac{\sum_{(i)\in U} \frac{Recommanded_i \cap Viewed_i}{Viewed_i}}{|U|} \tag{1.1}$$

Cet indicateur est relatif à une période. Nous l'avons fixé à 30 jours.

Par ailleurs, nous calculons arbitrairement cet indicateur pour les 10 plus grands utilisateurs (c'est-à-dire, les utilisateurs qui visionnent le plus de vidéo durant les 30 derniers jours).

Cet indicateur représente un ratio de vidéos potentiellement recommandées par un algorithmes parmi l'ensemble des vidéos effectivement visionnées par les plus grands utilisateurs.

Ce module est constitué de deux parties :

1. **Tracking :** (automatique) calcul et stockage quotidien des recommandations par chaque algorithme pour les 10 plus grands utilisateurs + collecte des vidéos quotidiennement visitées par ces mêmes utilisateurs.
2. **Monitoring :** (à la demande) calcul et visualisation graphique des indicateurs par algorithme.

Les algorithmes de recommandation concernés sont : Most Recent, Most Popular, CF Nearby, CF User Based, et CB User Based puisqu'ils sont les seuls algorithmes qui dépendent uniquement de l'utilisateur.

Conclusion

Avant de poursuivre sur les procédures détaillées d'installation de chaque module, nous devons souligner que l'architecture modulaire s'appuie concrètement sur des méthodes et des outils techniques de l'état de l'art. En effet, chaque module est déployée sous la forme d'une pile de conteneurs basés sur les technologies **Docker** et **Docker-compose** ; ce qui va faciliter le déploiement car les distributions des modules sont autonomes, c'est-à-dire, qu'elles ne nécessitent pas d'installation préalables de logiciels complexes et que l'installateur n'a pas à se soucier des conflits de version. La seule condition est d'avoir une machine où sont installés Docker et docker-compose (sous Unix, Windows ou MacOs). Le langage de programme utilisé pour le développement est le langage **Python 3**.

De plus, afin de simplifier la tâche de l'installateur, les principales commandes sont accessibles via la commande **make** associé à un fichier **Makefile**.

Par ailleurs, l'approche de développement choisie repose résolument sur le principe d'intégration continue (CI/CD). Ainsi, toutes les distributions sources des modules sont localisées dans un même groupe **gitlab**. Le serveur gitlab utilisé pour le projet Smart Vidéos est : `https://gitlab.inria.fr/noozy-ai`

Il est donc nécessaire d'avoir un compte sur ce serveur gitlab [1] pour effectuer les opérations d'installation présentées dans la suite de document.

1. La demande de compte doit être faite par mail à azim.roussanaly@loria.fr

Chapitre 2

Module Serveur xAPI

Le serveur xAPI permet la collecte des traces d'usages de la plateforme Noozy. Celui de NoozyIA est **TRAX-LRS V1**. Il est basé sur le standard xAPI [1]. Ainsi, toute interaction est envoyée comme étant une activité de type «acteur, verbe, objet» avec un timestamp, contexte, etc.
Cette activité est appelée **statement**. Elle est représentée par un objet JSON qui est stocké dans une base de données **MySQL** incluse dans TRAX-LRS. Pour plus de détails sur la structure des statements, nous renvoyons à [5].

Afin de généraliser la stratégie de d'intégration continue, nous avons créé une version dockerisable de TRAX-LRS.

Il est à noter que NoozyIA est indépendant du choix du serveur xAPI. Autrement dit, n'importe quel autre serveur conforme au standard xAPI du marché peut remplacer le serveur TRAX-LRS v1 proposé ici.

2.1 Prérequis

L'installation de TRAX-LRS requiert la disponibilité sur votre machine des commande suivantes :

- **git** : système de contrôle de version
- **docker** : système d'exploitation pour conteneurs
- **docker-compose** : système d'orchestration de conteneurs
- **make** : utilitaire d'installation
- **vi** : éditeur de texte

Il est aussi indispensable d'avoir un compte sur le serveur gitlab de l'INRIA (`https://gitlab.inria.fr/`)

2.2 Importation de l'image docker de TRAX-LRS

Cette étape est nécessaire car, il n'existe pas d'image docker de TRAX-LRS dans le **Hub Docker** publique. Une version de l'image est disponible dans le hub privé : **registry.gitlab.inria.fr**.

Cette étape n'est toutefois pas nécessaire en cas de réinstallation du serveur xAPI.

Les commandes :

```
$ docker login registry.gitlab.inria.fr
...
$ docker pull registry.gitlab.inria.fr/sisr/docker/trax/
   trax_trax
$ docker tag registry.gitlab.inria.fr/sisr/docker/trax/
   trax_trax trax_trax:latest
$ docker logout registry.gitlab.inria.fr
```

2.3 Téléchargement

```
$ mkdir install
$ cd install
$ git clone https://gitlab.inria.fr/noozy-ai/xapi4noozy.git
```

2.4 Structure et arborescence

Ce projet contient les scripts nécessaires pour l'installation du serveur TRAX-LRS (V1) et le démarrage du service xAPI dans un conteneur Docker.

Voici la liste des fichiers les plus importants de la distribution source :

- **docker-compose.yml :** fichier utilisé par la commande **docker-compose**
- **.env :** contient les paramètres à fixer avant l'installation.
- **Makefile :** regroupe les commandes « make » de gestion des conteneurs
- **build** contient les fichiers de configuration du serveur http et php ainsi que le `Dockerfile` contenant le code de création de l'image Trax.
- **scripts** contient les scripts d'utilitaires

- **src** contient les modules et les scripts en Python

2.5 Configuration

La configuration s'effectue en éditant le fichier **.env**

```
# Use 'always' in production
RESTART_SERVICES=no
# Timezone definition
TZ=Europe/Paris
# Listen to localhost only if an nginx proxy is in place
LISTEN_ITF=0.0.0.0

# PORT MAPPING
PORT_MYSQL=3306
PORT_PHPMYADMIN=8080
PORT_TRAX_HTTP=80
PORT_TRAX_HTTPS=443

#database conf
MYSQL_DIR=${HOME}/traxdata
MYSQL_ROOT_PASSWORD=****
MYSQL_USER=user
MYSQL_PASSWORD=****
```

Les paramètres à configurer :

- **PORT_TRAX_HTTP** : numéro de port pour accéder au serveur xAPI
- **MYSQL_DIR** : dossier de persistance de la base de données MySQL. Ce dossier sera créé sur la machine locale et contiendra le volume persistant utilisé par MySQL afin de récupérer la base de données en cas de panne et/ou de réinstallation.
- **MYSQL_ROOT_PASSWORD** : mot de passe root du SGBD MySQL
- **MYSQL_USER** : identifiant d'un utilisateur de la bases de donnée de Trax
- **MYSQL_PASSWORD** : mot de passe de l'utilisateur

2.6 Installation

```
$ cd xapi4noozy
$ make build
...
$ make up
...
$ make logs
...
$ make trax_initdb
$ make trax_create_admin
...
$ make trax_create_client
$
```

Les commandes correspondent respectivement à :

- **make build** : création de l'image docker de Trax-LRS
- **make up** : création et activation des conteneurs
- **make logs** : affichage de logs de démarrage des services : trax_db (MySQL), trax_pma (PhpMyAdmin), trax_xapi(serveur xAPI)
- **trax_initdb** : initialisation de la base de données. Attention, cette commande efface la base de données existante si elle existe déjà !
- **trax_create_admin** : création d'un utilisateur xAPI admin. La commande attribue automatiquement un mot de passe admin ; **n'oubliez pas de le noter lorsqu'il s'affiche !**
- **trax_create_client** : création d'un utilisateur xAPI client.

On peut vérifier l'activation des conteneurs avec la commande suivante :

```
$ make ps
 Name                 Command               State                 Ports
-------------------------------------------------------------------------------------
trax_db      docker-entrypoint.sh --def ...   Up      0.0.0.0:3306->3306/tcp, 33060/tcp
trax_pma     /docker-entrypoint.sh apac ...   Up      0.0.0.0:8080->80/tcp
$
```

Il est aussi possible de tester le démarrage du serveur xAPI à l'aide d'un navigateur à l'URL : `http://<serveur>/`

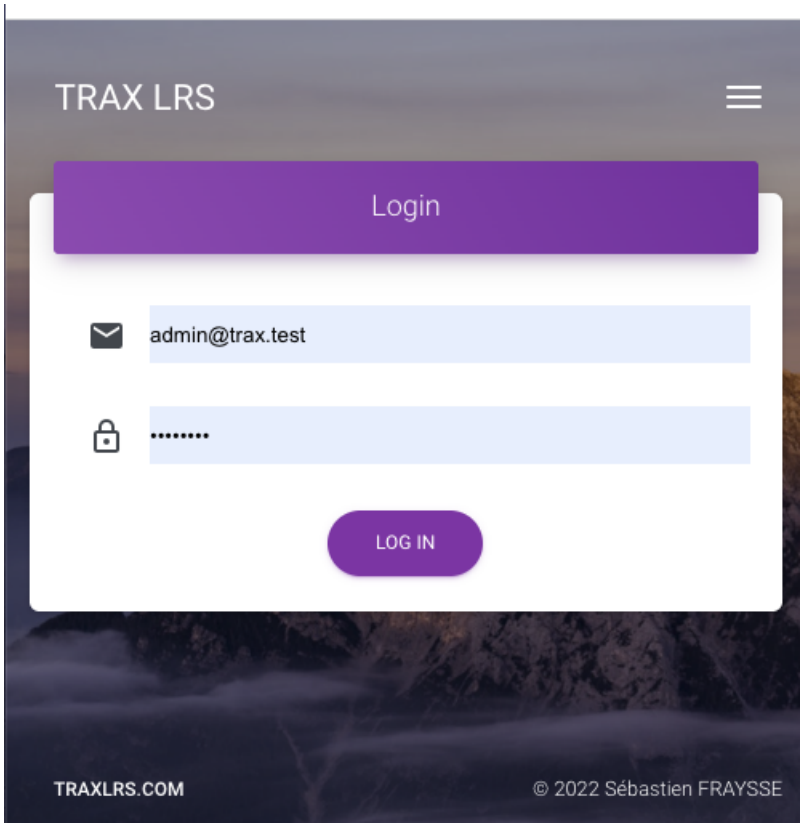

FIGURE 2.1 – Test serveur xAPI

Chapitre 3

Module Harvester

Ce module a pour objectif d'interroger le catalogue des vidéos disponibles et de collecter leurs métadonnées dans une base de données REDIS. Les requêtes s'effectuent via le standard **OAI-PMH** et les métadonnées sont transmises au format standard JSON **Dublin-Core**.

Ce module comporte deux conteneurs : 1) une base données REDIS qui permet de stocker les métadonnées associées à chaque identifiant de vidéo et 2) un scheduler qui met à jour les métadonnées en activant quotidiennement le Harvester.

3.1 Prérequis

L'installation du module Harvester requiert la disponibilité sur votre machine des commande suivantes :

- **git** : système de contrôle de version
- **docker** : système d'exploitation pour conteneurs
- **docker-compose** : système d'orchestration de conteneurs
- **make** : utilitaire d'installation
- **vi** : éditeur de texte

Il est aussi indispensable d'avoir un compte sur le serveur gitlab de l'INRIA (`https://gitlab.inria.fr/`)

3.2 Téléchargement

```
$ mkdir install
$ cd install
$ git clone https://gitlab.inria.fr/noozy-ai/harvest4noozy.
   git
```

3.3 Structure et arborescence

Le projet contient les scripts nécessaires pour installer la base de données et exécuter le harvester. Les fichiers les plus importants sont :

- **docker-compose.yml :** fichier utilisé par la commande **docker-compose**
- **.env :** contient les paramètres à fixer avant l'installation.
- **Makefile :** regroupe les commandes « make » de gestion des conteneurs
- **redis** contient les fichiers de configuration du serveur redis ainsi que le `Dockerfile` contenant le code de création de l'image Redis.
- **harvest/src/*.PY :** contiennent les modules et les scripts en Python
- **harvest/src/noozy.ini :** contient les paramètres de configuration

3.4 Configuration

La configuration s'effectue en éditant le fichier **.env** et le fichier **harvest/src/-noozy.ini**

Le fichier .env

```
# This is the docker-compose environment file
#Port
PORT=6381
#Password (must be changed for security reason)
PASSWORD=****
```

Les paramètres à configurer :

- **PORT** : numéro de port pour accéder au serveur REDIS des métadonnées. NoozyIA utilise plusieurs serveurs REDIS. Nous recommandons le port 6381 pour celui-ci.
- **PASSWORD** : mot de passe pour écrire dans SGBD REDIS.

Le fichier harvest/src/noozy.ini

```
[DEFAULT]
LastDateStoreFileName = ./.former_harvest_date
LogFileName = ./noozy.log

[OAI]
Host = noozy.tv
User = client
Password = ****
Port = 80

[REDIS]
Host = redis
Port = 6381
Password = ****
```

Les paramètres à configurer :

- **OAI.Host** : hôte où se trouve le serveur OAI/PMH.
- **OAI.User** : identifiant de l'utilisateur OAI
- **OAI.Password** : mot de passe de l'utilisateur OAI
- **OAI.Port** : numéro de port du serveur OAI/PMH
- **REDIS.Host** : hôte ou conteneur du serveur REDIS.
- **REDIS.Port** : numéro de port du serveur REDIS (métadonnées)
- **REDIS.Password** : mot de passe REDIS (le même que celui du fichier .env)

3.5 Installation

```
$ cd xapi4noozy
$ make build
...
$ make up
...
$ make logs
...
```

```
$ make harv
...
$
```

Les commandes correspondent respectivement à :

- **make build** : création de l'image docker de Trax-LRS
- **make up** : création et activation des conteneurs
- **make logs** : affichage de logs de démarrage des services : redis_harvest, harvest
- **make harv** : démarrer une première collecte de métadonnées (sachant que les collectes suivantes sont planifiées quotidiennement à 01 :00.

Chapitre 4

Modules Algorithmes

NoozyIA comprend plusieurs modules Algorithmes. Ceux-ci son présentés dans le chapitre 1. Dans les futures versions, d'autres algorithmes sont susceptibles d'enrichir la collection existante.

Leur conception selon un modèle standardisé, facilite d'une part, l'adjonction de nouveaux algorithmes et, rend d'autre part, leur procédure d'installation très similaire. Par conséquent, nous nous contenterons de faire une seule présentation pour tous les modules Algorithmes.

Les algorithmes calculent quotidiennement les recommandations et les stockent dans deux bases de données REDIS différentes. La première est utilisée par les algorithmes dits **item-based**, c'est-à-dire qu'une liste de recommandations est calculée pour chaque vidéo (autrement dit, la clef d'accès est un identifiant de vidéo). La seconde est destinée aux algorithmes dits **user-based**, c'est-à-dire qu'une liste de recommandations est calculée pour chaque utilisateur (autrement dit, la clef d'accès est un identifiant d'utilisateur).

Ces bases de données sont alimentées par les différents algorithmes et sont consultés par le **Module Moteur de recommandations** que nous allons présenter dans le chapitre suivant.

Il est donc nécessaire d'installer préalablement un module supplémentaire qui met en place ces deux bases de données.

4.1 Module Redis4Engine

4.1.1 Prérequis

L'installation du module Redis4Engine requiert la disponibilité sur votre machine des commande suivantes :

- **git** : système de contrôle de version
- **docker** : système d'exploitation pour conteneurs
- **docker-compose** : système d'orchestration de conteneurs
- **make** : utilitaire d'installation
- **vi** : éditeur de texte

Il est aussi indispensable d'avoir un compte sur le serveur gitlab de l'INRIA (`https://gitlab.inria.fr/`)

4.1.2 Téléchargement

```
$ cd install
$ git clone https://gitlab.inria.fr/noozy-ai/redis4engine.
   git
```

4.1.3 Structure et arborescence

Le projet contient les scripts nécessaires pour installer la base de données et démarrer les deux bases de données REDIS partagées par les Modules Algorithmes et le Module Moteur de recommandations. Les fichiers les plus importants sont :

- **docker-compose.yml :** fichier utilisé par la commande **docker-compose**
- **.env :** contient les paramètres à fixer avant l'installation.
- **Makefile :** regroupe les commandes « make » de gestion des conteneurs
- **redis1** contient les fichiers de configuration du premier serveur redis ainsi que le `Dockerfile` contenant le code de création de l'image Redis.
- **redis2** contient les fichiers de configuration du second serveur redis ainsi que le `Dockerfile` contenant le code de création de l'image Redis.

4.1.4 Configuration

La configuration s'effectue en éditant le fichier **.env**

Le fichier .env

```
# This is the docker-compose environment file
```

```
# Since we are launching two redis services from the same
   project,
# We will specify two different ports and we will keep the
   same password
#Ports
PORT1=6380
PORT2=6382
#Password (must be changed for security reason)
PASSWORD=****
```

Les paramètres à configurer :

- **PORT1** : numéro de port pour accéder au serveur REDIS#1 des recommandations (user-based). (recommandé : 6380)
- **PORT2** : numéro de port pour accéder au serveur REDIS#2 des recommandations (item-based). (recommandé : 6382).
- **PASSWORD** : mot de passe pour écrire dans les SGBD REDIS (le même pour les deux serveurs)

4.1.5 Installation

```
$ cd redis4engine
$ make build
...
$ make up
...
$ make logs
...

$
```

Les commandes correspondent respectivement à :

- **make build** : création des images docker de REDIS
- **make up** : création et activation des conteneurs
- **make logs** : affichage de logs de démarrage des services : engine-redis1 , engine-redis2

4.2 Modules Algorithmes

La liste des 11 algorithmes est fournie à la fin du chapitre 1.

Au niveau de leur distribution source sur **gitlab**, nous les avons divisés en deux parties chacun :

- la branche **dev** consacrée au développement, dépouillée de tous les artifices destinés à la procédure de déploiement ; elle peut être téléchargée, modifiée, testée et mise à jour par un développeur en mode **standalone**,
- la branche **install** qui fait référence au précédent et qui ne contient que le code de paramétrisation et de déploiement de l'algorithme.

Dans ce chapitre, nous nous intéressons exclusivement à l'installation, c'est-à-dire à la branche install. C'est le processus d'installation qui s'occupe de télécharger automatiquement la branche dev correspondante.

Dans la suite de la présentation, nous allons arbitrairement fournir les commandes concernant le module **Most Popular**. La procédure pourra être répétée pour les 10 autres modules Algorithmes.

4.2.1 Téléchargement

```
$ cd install
$ git clone --branch MostPopular-install https://gitlab.inria.fr/noozy-ai/most-popular-algorithm.git most-
    popular-algorithm
```

4.2.2 Structure et arborescence

Le projet contient les scripts nécessaires pour récupérer les programmes sources de l'algorithme dans le **gitlab** et pour planifier les activations quotidiennes de l'algorithme. Les fichiers les plus importants sont :

- **docker-compose.yml :** fichier utilisé par la commande **docker-compose**
- **.env :** NB : les paramètres définis dans le fichier .env ne sont pas pris en compte dans cette version.
- **Makefile :** regroupe les commandes « make » de gestion des conteneurs
- **noozy.ini :** paramètres externes à configurer avant l'installation
- **recommender_algorithm_input.yml :** paramètres internes à configurer avant l'installation

4.2.3 Configuration

La configuration s'effectue en éditant les fichiers **noozy.ini** et **recommender_algorithm_input.yml**

Le fichier noozy.ini

```
[DEFAULT]
LogFileName = ./noozy.log
[REDIS]
Host = loria.kd-serveur.com
Password = ****
Port = 6381
[STATEMENTS]
Host = loria.kd-serveur.com
User = testsuite
Password = ****
Port = 80
[RESULT_REDIS]
Host = loria.kd-serveur.com
Password = ****
Port = 6380
```

Les paramètres à configurer :

- **REDIS.Host** : nom du serveur où est installé la base de données REDIS des métadonnées (voir Module Harvester)
- **REDIS.Port** : numéro de port d'écoute du serveur REDIS des métadonnées
- **REDIS.Password** : mot de passe du serveur REDIS des métadonnées
- **STATEMENTS.Host** : nom du serveur où est installé TRAX-LRS (voir Module xAPI)
- **STATEMENTS.User** : nom de l'utilisateur xAPI
- **STATEMENTS.Password** : mot de passe de l'utilisateur xAPI
- **RESULT_REDIS.Host** : nom du serveur où est installé la base de données REDIS de stockage des recommandations (voir Module Redis4Engine)
- **RESULT_REDIS.Port** : numéro de port d'écoute du serveur REDIS de stockage des recommandations (ici on choisira le port dédié au REDIS user-based)

- **RESULT_REDIS.Password** : mot de passe du serveur REDIS de stockage des recommandations

Le fichier recommender_algorithm_input.yml

```
input:
  description:
    - "algorithm_id: required"
    - "n: required"
    - "user_id_requirement: 'optional'(default)"
    - "video_id_requirement: 'required' / 'optional' / 'not required' (default)"
    - "dependant_data_parameters - a list of needed parameters for analysis (required)"
  # name of the algorithm module (./src/algorithm/most_popular.py)
  algorithm_id : "most_popular"
  n: 20
  # for all algorithm, user_id should be 'optional'... this is not actually needed
  user_id_requirement: "optional"
  # video_id input - only required for content based / item based algorithm
  video_id_requirement: "not required"
  # dependant_data_parameters (eg. 'dc:description' OR 'dc:available') 'dc:identifier' - default
  dependant_data_parameters : ['dc:identifier', 'dc:title', 'dc:description', 'dc:type', 'dc:subject', 'dc:
      available', 'dc:source.title']
  date_range_dependency: "False"
  date_range_before: 30
  date_range_after: 30
  start_from :
    day: 1
    month: 1
    year: 2021
```

Les paramètres de l'algorithme à configurer si besoin :

- **n** : nombre maximal de recommandations à prévoir
- **start_from** : date de début pour la sélection des données à utiliser (par exemple, les plus populaires depuis 1/1/2021

4.2.4 Installation

```
$ cd ost-popular-algorithm
$ make build
...
$ make up
...
$ make logs
...

$
```

Les commandes correspondent respectivement à :

- **make build** : création des images docker de REDIS
- **make up** : création et activation des conteneurs
- **make logs** : affichage de logs de démarrage du service : most-popular-algorithm

4.3 Généralisation aux autres modules Algorithmes

Le tableau 4.1 fournit les informations nécessaires à l'adaptation de la procédure d'installation pour les autres algorithmes.

Algorithmes	Paramètres
Most Recent	identifiant : most_recent url : https://gitlab.inria.fr/noozy-ai/most-recent-algorithm.git branche : MostRecent-install type : user-based
Most Popular	identifiant : most_popular url : https://gitlab.inria.fr/noozy-ai/most-popular-algorithm.git branche : MostPopular-install type : user-based
Popular by Genre	identifiant : popular_by_genre url :https://gitlab.inria.fr/noozy-ai/popular-by-genre-algorithm.git branche : install type : user-based
Popular by Contributor	identifiant : popular_by_contributor url :https://gitlab.inria.fr/noozy-ai/popular-by-contributor-algorithm.git branche : install type : user-based
Popular by Thematic	identifiant : popular_by_thematic url :https://gitlab.inria.fr/noozy-ai/popular-by-thematic-algorithm.git branche : install type : user-based
CF Nearby	identifiant : cf_nearby url :https://gitlab.inria.fr/noozy-ai/collaborative-filter-algorithm.git branche : CF-Nearby-install type : user-based
CF User Based	identifiant : cf_userbased url : https://gitlab.inria.fr/noozy-ai/collaborative-filter-algorithm.git branche : CF-UserBased-install type : user-based
CF Item Based	identifiant : cf_itembased url :https://gitlab.inria.fr/noozy-ai/collaborative-filter-algorithm.git branche : CF-ItemBased-install type : item-based
CB User Based	identifiant : CB_UserBased url :https://gitlab.inria.fr/noozy-ai/content-based-algorithm.git branche : CB-UserBased-install type : user-based
CB Item Based	identifiant : CB_ItemBased url : https://gitlab.inria.fr/noozy-ai/content-based-algorithm.git branche : CB-ItemBased-install type : item-based
Random	identifiant : random_basic url :https://gitlab.inria.fr/noozy-ai/random-algorithm.git branche : Random-install type : user-based

TABLE 4.1 – Paramètres à adapter pour chaque module Algorithme

Chapitre 5

Module Moteur de recommandations

Ce module se présente comme une API RESTful qui permet de traiter des requêtes de demande de recommandations. Pour ce faire, il s'appuie sur les 2 bases REDIS du module **Redis4Engine** (voir section 4.1).

Remarque : il est donc nécessaire que le module **Redis4Engine** soit préalablement opérationnel.

Le module **Moteur de recommandations** est implémenté comme un serveur **Python/Flask**. Voici la liste complète des services exposés (ou **endpoints**) au format YAML/OpenApi [2].

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Recommender Engine
  description: >-
    This API serves to request the server to get the recommendation results for Noozy.tv platform.
  license:
    name: lgpl3 Licence

servers:
  - description: The url presents the host of the remote server and the port where the application is running
    url: http://loria.kd-serveur.com:8088

paths:

  /Random:
    get:
      tags:
        - Random Algorithm
      description: The algorithm shuffle randomly and return a list of top N videos from the catalogue that
           are not seen by the user
      operationId: getRandomVideosbyUser
      parameters:
        - name: n
          in: query
          description: Number of recommended videos to return
          required: true
          schema:
            type: integer
            format: int64
        - name: user_id
          in: query
          description: ID of the current user, None if the user is Unknown
```

```yaml
        required: false
        schema:
          type: string

    responses:
      '200':
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Result'
      '400':
        description: Invalid user ID supplied
        content: {}

/MostRecent:
  get:
    tags:
      - Recent Algorithms
    description: The algorithm sorts the videos and returns a list of top N recently added videos not seen
         by the user
    operationId: getMostRecentVideosbyUser
    parameters:
      - name: n
        in: query
        description: Number of recommended videos to return
        required: true
        schema:
          type: integer
          format: int64
      - name: user_id
        in: query
        description: ID of the current user, None if the user is Unknown
        required: false
        schema:
          type: string

    responses:
      '200':
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Result'
      '400':
        description: Invalid user ID supplied
        content: {}

/Recent-By-Genre:
  get:
    tags:
      - Recent Algorithms
    description: The algorithm sorts the videos and returns a list of top N recently added videos having
         the same input genre and not yet seen by the user
    operationId: getMostRecentVideosByGenre
    parameters:
      - name: n
        in: query
        description: Number of recommended videos to return
        required: true
        schema:
          type: integer
          format: int64
      - name: user_id
        in: query
        description: ID of the current user, None if the user is Unknown
        required: false
        schema:
          type: string
      - name: genre
        in: query
        description: Genre Name
        required: true
        schema:
          type: string

    responses:
      '200':
        description: Successful operation
```

```
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Result'
      '400':
        description: Invalid user ID supplied
        content: {}


/Recent-By-Thematic:
  get:
    tags:
      - Recent Algorithms
    description: The algorithm sorts the videos and returns a list of top N recently added videos having
         the same input thematic and not yet  not seen by the user
    operationId: getMostRecentVideosbyThematic
    parameters:
      - name: n
        in: query
        description: Number of recommended videos to return
        required: true
        schema:
          type: integer
          format: int64
      - name: user_id
        in: query
        description: ID of the current user, None if the user is Unknown
        required: false
        schema:
          type: string
      - name: thematic
        in: query
        description: Thematic Name
        required: true
        schema:
          type: string

    responses:
      '200':
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Result'
      '400':
        description: Invalid user ID supplied
        content: {}


/Recent-By-Contributor:
  get:
    tags:
      - Recent Algorithms
    description: The algorithm sorts the videos and returns a list of top N recently added videos having
         the same input contributor and not yet not seen by the user
    operationId: getMostRecentVideosbyContributor
    parameters:
      - name: n
        in: query
        description: Number of recommended videos to return
        required: true
        schema:
          type: integer
          format: int64
      - name: user_id
        in: query
        description: ID of the current user, None if the user is Unknown
        required: false
        schema:
          type: string
      - name: contributor
        in: query
        description: Contributor Name
        required: true
        schema:
          type: string

    responses:
      '200':
```

```
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Result'
      '400':
        description: Invalid user ID supplied
        content: {}

/MostPopular:
  get:
    tags:
      - Popular Algorithms
    description: The algorithm sorts videos by their visualization frequency and returns the list of top N
         visualized videos not seen by the user is exists
    operationId: getMostpopularVideosByUser
    parameters:
      - name: n
        in: query
        description: Number of recommended videos to return
        required: true
        schema:
          type: integer
          format: int64
      - name: user_id
        in: query
        description: ID of the current user, None if the user is Unknown
        required: false
        schema:
          type: string

    responses:
      '200':
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Result'
      '400':
        description: Invalid user ID supplied
        content: {}

/Popular-By-Genre:
  get:
    tags:
      - Popular Algorithms
    description: The algorithm returns a list of top N most frequently visited videos having the same input
          genre and not yet seen by the user
    operationId: getMostPopularVideosByGenre
    parameters:
      - name: n
        in: query
        description: Number of recommended videos to return
        required: true
        schema:
          type: integer
          format: int64
      - name: user_id
        in: query
        description: ID of the current user, None if the user is Unknown
        required: false
        schema:
          type: string
      - name: genre
        in: query
        description: Genre Name
        required: true
        schema:
          type: string

    responses:
      '200':
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Result'
      '400':
        description: Invalid user ID supplied
```

```
        content: {}

/Popular-By-Thematic:
  get:
    tags:
      - Popular Algorithms
    description: The algorithm returns a list of top N most frequently visited videos having the same input
          thematic and not yet  not seen by the user
    operationId: getMostPopularVideosbyThematic
    parameters:
      - name: n
        in: query
        description: Number of recommended videos to return
        required: true
        schema:
          type: integer
          format: int64
      - name: user_id
        in: query
        description: ID of the current user, None if the user is Unknown
        required: false
        schema:
          type: string
      - name: thematic
        in: query
        description: Thematic Name
        required: true
        schema:
          type: string

    responses:
      '200':
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Result'
      '400':
        description: Invalid user ID supplied
        content: {}


/Popular-By-Contributor:
  get:
    tags:
      - Popular Algorithms
    description: The algorithm returns a list of top N most frequently visited videos having the same input
          contributor and not yet not seen by the user
    operationId: getMostPopularVideosbyContributor
    parameters:
      - name: n
        in: query
        description: Number of recommended videos to return
        required: true
        schema:
          type: integer
          format: int64
      - name: user_id
        in: query
        description: ID of the current user, None if the user is Unknown
        required: false
        schema:
          type: string
      - name: contributor
        in: query
        description: Contributor Name
        required: true
        schema:
          type: string

    responses:
      '200':
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Result'
      '400':
        description: Invalid user ID supplied
```

```
        content: {}


/CF-Nearby:
  get:
    tags:
      - Collaborative Filtering
    description: The algorithm  calculates the nearby users to the given user and returns a list of the top
          N videos visualized by his neighbors
    operationId: getVideosByCFNearby
    parameters:
      - name: n
        in: query
        description: Number of recommended videos to return
        required: true
        schema:
          type: integer
          format: int64
      - name: user_id
        in: query
        description: ID of the current user, None if the user is Unknown
        required: false
        schema:
          type: string

    responses:
      '200':
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Result'
      '400':
        description: Invalid user ID supplied
        content: {}


/CF-UserBased:
  get:
    tags:
      - Collaborative Filtering
    description: The algorithm calculates the nearest neighbors of the given user and returns a list of the
          top N videos visualized by his neighbors
    operationId: getVideosByCFUserBased
    parameters:
      - name: n
        in: query
        description: Number of recommended videos to return
        required: true
        schema:
          type: integer
          format: int64
      - name: user_id
        in: query
        description: ID of the current user, None if the user is Unknown
        required: false
        schema:
          type: string

    responses:
      '200':
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Result'
      '400':
        description: Invalid user ID supplied
        content: {}


/CF-ItemBased:
  get:
    tags:
      - Collaborative Filtering
    description: The algorithm calculates the nearest neighbors of the input video and returns a list of
         the top N similar videos not yet seen by the user
```

```
    operationId: getVideosByCFItemBased
    parameters:
      - name: n
        in: query
        description: Number of recommended videos to return
        required: true
        schema:
          type: integer
          format: int64
      - name: user_id
        in: query
        description: ID of the current user, None if the user is Unknown
        required: false
        schema:
          type: string
      - name: video_id
        in: query
        description: ID of the current video
        required: true
        schema:
          type: string

    responses:
      '200':
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Result'
      '400':
        description: Invalid ID(s) supplied
        content: {}

/CB-UserBased:
  get:
    tags:
      - Content Based
    description: The algorithm calculates the similarity between video contents and returns the top N most
        similar videos to the 3 last seen videos by the user
    operationId: getVideosByCBUserBased
    parameters:
      - name: n
        in: query
        description: Number of recommended videos to return
        required: true
        schema:
          type: integer
          format: int64
      - name: user_id
        in: query
        description: ID of the current user, None of the user is Unknown
        required: false
        schema:
          type: string

    responses:
      '200':
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Result'
      '400':
        description: Invalid user ID supplied
        content: {}


/CB-ItemBased:
  get:
    tags:
      - Content Based
    description: The algorithm calculates the similarity between video contents and returns the top N most
        similar videos to the input video seen by the user
    operationId: getVideosByCBItemBased
    parameters:
      - name: n
        in: query
        description: Number of recommended videos to return
```

```
          required: true
          schema:
            type: integer
            format: int64
        - name: user_id
          in: query
          description: ID of the current user, None if the user is Unknown
          required: false
          schema:
            type: string
        - name: video_id
          in: query
          description: ID of the current video
          required: true
          schema:
            type: string

      responses:
        '200':
          description: Successful operation
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Result'
        '400':
          description: Invalid ID(s) supplied
          content: {}


components:
  schemas:
    Result:
      type: object
      required:
        - video_id
        - algorithm_id
        - rank
        - score
      properties:
        video_id:
          description: The recommended video identifiant
          type: string
        algorithm_id:
          description: The used recommendation algorithm
          type: string
        rank:
          description: The Rank of the recommended video
          type: integer
        score:
          description: The score of the recommended video
          type: number

  securitySchemes:
    basicAuth:
      type: http
      scheme: basic

security:
  - basicAuth: []
```

Une documentation en ligne est aussi disponible ici : `https://gitlab.inria.fr/noozy-ai/recommender-engine/-/blob/dev/openapi.yaml`

5.1 Prérequis

L'installation du module Harvester requiert la disponibilité sur votre machine des commande suivantes :

- **git** : système de contrôle de version

- **docker** : système d'exploitation pour conteneurs
- **docker-compose** : système d'orchestration de conteneurs
- **make** : utilitaire d'installation
- **vi** : éditeur de texte

Il est aussi indispensable d'avoir un compte sur le serveur gitlab de l'INRIA (`https://gitlab.inria.fr/`)

5.2 Téléchargement

```
$ cd install
$ git clone https://gitlab.inria.fr/noozy-ai/recommender-
   engine.git
```

5.3 Structure et arborescence

Le projet contient les scripts nécessaires pour installer la base de données et exécuter le moteur de recommandations. Les fichiers les plus importants sont :

- **docker-compose.yml :** fichier utilisé par la commande **docker-compose**
- **Makefile :** regroupe les commandes « make » de gestion des conteneurs
- **openapi.yaml :** fichier décrivant les services
- **swagger_server :** code source du serveur Flask
- **swagger_server/controllers/noozy.ini :** contient les paramètres de configuration

5.4 Configuration

La configuration s'effectue en éditant le fichier **swagger_server/controllers/noozy.ini**

Le fichier swagger_server/controllers/noozy.ini

```
[REDIS]
Host = loria.kd-serveur.com
Password = ****
```

```
Port = 6381

[STATEMENTS]
Host = loria.kd-serveur.com/trax/ws/xapi/statements
User = ****
Password = ****

[RESULT_REDIS1]
Host = loria.kd-serveur.com
Password = ****
Port = 6380

[RESULT_REDIS2]
Host = loria.kd-serveur.com
Password = ****
Port = 6382
```

Les paramètres à configurer :

- **REDIS.Host** : hôte où se trouve la base REDIS des métadonnées.
- **REDIS.Port** : numéro de port du serveur REDIS (métadonnées)
- **REDIS.Password** : mot de passe REDIS (métadonnées)
- **STATEMENTS.Host** : hôte où se trouve le serveur xAPI.
- **STATEMENTS.Port** : numéro de port du serveur xAPI (80 par défaut)
- **STATEMENTS.User** : identifiant utilisateur xAPI
- **STATEMENTS.Password** : mot de passe
- **RESULT_REDIS1.Host** : hôte où se trouve la base REDIS des derniers résultats calculés par les algorithmes (user-based).
- **RESULT_REDIS1.Port** : numéro de port du serveur REDIS (user-based)
- **RESULT_REDIS1.Password** : mot de passe REDIS (user-based)
- **RESULT_REDIS2.Host** : hôte où se trouve la base REDIS (item-based).
- **RESULT_REDIS2.Port** : numéro de port du serveur REDIS (item-based)
- **RESULT_REDIS2.Password** : mot de passe REDIS (item-based)

5.5 Installation

```
$ cd xapi4noozy
$ make build
...
```

```
$ make up
...
$ make logs
...
$
```

Remarques : dans la version actuelle, le moteur de recommandation utilise le **port 8088**. Il n'est pas possible de le paramétrer ! Par ailleurs, le serveur Flask utilise le même identifiant et le même mot de passe que le serveur xAPI.

Chapitre 6

Modules Tracking et Monitoring

Afin de suivre l'évolution des performances des principaux algorithmes (Most Recent, Most Popular, CF Nearby, CF User Based, et CB User Based), nous proposons un outil formé de trois modules :

- le module **Redis4Analytics** qui est un module qui crée une base de données REDIS dans laquelle seront stockées les données collectées par le module **Tracking**,
- le module **Tracking** qui permet de collecter régulièrement des données,
- le module **Monitoring** qui utilise les données collectées par le Tracking pour calculer un indicateur de performance : la couverture, définie à la section 1.5.

Prérequis L'installation de ces deux module requiert la disponibilité sur votre machine des commande suivantes :

- **git** : système de contrôle de version
- **docker** : système d'exploitation pour conteneurs
- **docker-compose** : système d'orchestration de conteneurs
- **make** : utilitaire d'installation
- **vi** : éditeur de texte

Il est aussi indispensable d'avoir un compte sur le serveur gitlab de l'INRIA (`https://gitlab.inria.fr/`)

6.1 Module Redis4Analytics

6.1.1 Téléchargement

```
$ cd install
$ git clone https://gitlab.inria.fr/noozy-ai/redis4analytics
   .git
```

6.1.2 Structure et arborescence

Le projet contient les scripts nécessaires pour installer le module. Les fichiers les plus importants sont :

- **docker-compose.yml :** fichier utilisé par la commande **docker-compose**
- **.env :** contient les paramètres à fixer avant l'installation.
- **Makefile :** regroupe les commandes « make » de gestion des conteneurs
- **redis** contient les fichiers de configuration du serveur redis ainsi que le `Dockerfile` contenant le code de création de l'image Redis.
- **test/*.py :** contiennent les scripts en Python

6.1.3 Configuration

La configuration s'effectue en éditant le fichier **.env**

Le fichier .env

```
# This is the docker-compose environment file
# Since we are launching two redis services from the same
   project,
# We will specify two different ports and we will keep the
   same password
# Ports
PORT=6379
# Password (must be changed for security reason)
PASSWORD=****
# database conf
REDIS_DIR=${HOME}/redis-data
```

Les paramètres à configurer :

- **PORT** : numéro de port du serveur REDIS (pour le tracking) (recommandé : 6379)

- **PASSWORD** : mot de passe pour écrire dans les SGBD REDIS
- **REDIS_DIR** : volume de persistance de la base données (en cas de redémarrage du service)

6.1.4 Installation

```
$ cd redis4engine
$ make build
...
$ make up
...
$ make logs
...

$
```

Les commandes correspondent respectivement à :

- **make build** : création de l'image docker de REDIS
- **make up** : création et activation des conteneurs
- **make logs** : affichage de logs de démarrage des services : analytics-utils et analytics-redis

6.2 Module Tracking

Le principe du tracking consiste à sélectionner quotidiennement 10 utilisateurs (en l'occurrence les utilisateurs qui ont visionné le plus de vidéo durant les 30 derniers jours) et, pour chacun d'entre eux, à stocker d'une part, les vidéos réellement visionnées et, d'autre part, les vidéos (théoriquement) recommandées par les algorithmes monitorés.

Les résultats sont stockés dans la base de données REDIS créée dans le module **Redis4Analytics**.

6.2.1 Téléchargement

```
$ cd install
```

```
$ git clone https://gitlab.inria.fr/noozy-ai/tracking4noozy.
   git
```

6.2.2 Structure et arborescence

Le projet contient les scripts nécessaires pour installer le module. Les fichiers les plus importants sont :

- **docker-compose.yml :** fichier utilisé par la commande **docker-compose**
- **Makefile :** regroupe les commandes « make » de gestion des conteneurs
- **src** contient les fichiers sources en Python
- **noozy.ini :** contient les paramètres de configuration

6.2.3 Configuration

La configuration s'effectue en éditant le fichier **noozy.ini**

Le fichier noozy.ini

```
[ENGINE]
Host = loria.kd-serveur.com:8088
User = testsuite
Password = ****
[ANALYTICS_REDIS]
Host = ****
Password = ****
Port = 6379
```

Les paramètres à configurer :

- **ENGINE.Host** : URL du service Moteur de recommandations (
- **STATEMENTS.User** : identifiant utilisateur correspondant
- **STATEMENTS.Password** : mot de passe correspondant
- **ANALYTICS_REDIS.Host** : hôte où doivent être stockées les données de tracking
- **ANALYTICS_REDIS.Password** : mot de passe correspondant
- **ANALYTICS**$_R$*EDIS.Port* : *portcorrespondant*

6.2.4 Installation

```
$ cd tracking4noozy
$ make build
...
$ make up
...
$ make logs
...

$
```

Les commandes correspondent respectivement à :

- **make build** : création de l'image docker de REDIS
- **make up** : création et activation des conteneurs
- **make logs** : affichage de logs de démarrage du service : tracking

6.3 Module Monitoring

Le monitoring consiste à calculer un indicateur de performance pour chaque algorithme. L'indice retenu est ici la couverture (voir section 1.5). Cet indice traduit le pourcentage de vidéos recommandées qui ont été effectivement visualisées par les utilisateurs. L'indice est calculé sur une période des 7 jours précédents pour les 10 utilisateurs les plus actifs sur la plateforme.

Le principe de fonctionnement du monitoring est double :

1. **automatique** : il est possible d'activer un conteneur qui calcule quotidiennement les indicateurs pour chaque algorithme et qui les stocke dans un dossier persistant (par exemple $HOME/dashboard),
2. **interactif** : il est aussi possible d'utiliser le module comme une application en mode ligne de commande pour générer un graphique comparatif.

6.3.1 Téléchargement

```
$ cd install
```

```
$ git clone https://gitlab.inria.fr/noozy-ai/
   monitoring4noozy.git
```

6.3.2 Structure et arborescence

Le projet contient les scripts nécessaires pour installer le module. Les fichiers les plus importants sont :

- **docker-compose.yml :** fichier utilisé par la commande **docker-compose**
- **Makefile :** regroupe les commandes « make » de gestion des conteneurs
- **src** contient les fichiers sources en Python
- **noozy.ini :** contient les paramètres de configuration du monitoring
- **.env :** contient les paramètres de configuration de l'installation

6.3.3 Configuration

Pour configurer l'installation, on utilise le fichier **.env**.

Le fichier .env

```
DASHBOARD\_DIR=${HOME}/noozy-dashboard
```

Les paramètres à configurer :

- **DASHBOARD_DIR** : nom du dossier où on va stocker les indices calculés quotidiennement

Pour configurer l'application, on utilise le fichier **noozy.ini**

Le fichier noozy.ini

```
#config file for monitoring
#NoozyIA project
#---------------------------
[ANALYTICS_REDIS]
Host = ****
Password = ****
Port = 6379
```

Les paramètres à configurer :

- **ANALYTICS_REDIS.Host** : hôte où doivent être stockées les données de tracking
- **ANALYTICS_REDIS.Password** : mot de passe correspondant
- **ANALYTICS_REDIS.Port** : port correspondant

6.3.4 Installation

Nous avons évoqué précédemment les deux principes de fonctionnement du monitoring : automatique et interactif.

Pour le mode automatique, on active un conteneur.

```
$ cd monitoring4noozy
$ make build
...
$ make up
...
$ make logs
...

$
```

Les commandes correspondent respectivement à :

- **make build** : création de l'image docker de REDIS
- **make up** : création et activation des conteneurs
- **make logs** : affichage de logs de démarrage du service : tracking

Pour le mode interactif, on utilise la ligne de commande dans un terminal. Cela nécessite d'avoir installé préalablement un interpréteur **python3**.

Pour voir tous les arguments de la ligne de commande :

```
Application de monitoring : NoozyIA'
      Created by Azim Roussanaly, Olfa Messaoud and Asif
         Ahmed
      Copyright 2021 - LORIA/Universit  de Lorraine. All
         rights reserved.

      Licensed under the Apache License 3.0
```

```
      http://www.apache.org/licenses/LICENSE-3.0

      Distributed on an "AS IS" basis without warranties
      or conditions of any kind, either express or implied.

optional arguments:
  -h, --help            show this help message and exit
  -c filename, --config filename
                        .ini file
  -i, --interactive     interactive mode on
  -a iso date, --date iso date
                        iso format date
  -d, --debug           debug mode on
```

Pour voir les indicateurs à une date donnée :

```
$ python src/app.py --date 2022-2-8 --interactive
```

On obtient une fenêtre graphique (voir figure 6.1)

On peut aussi obtenir le même graphique directement dans un fichier du dossier **dashboard**.

```
$ python src/app.py --date 2022-2-8
$ ls dashboard
noozy-coverage-plot-2022-01-01.png
```

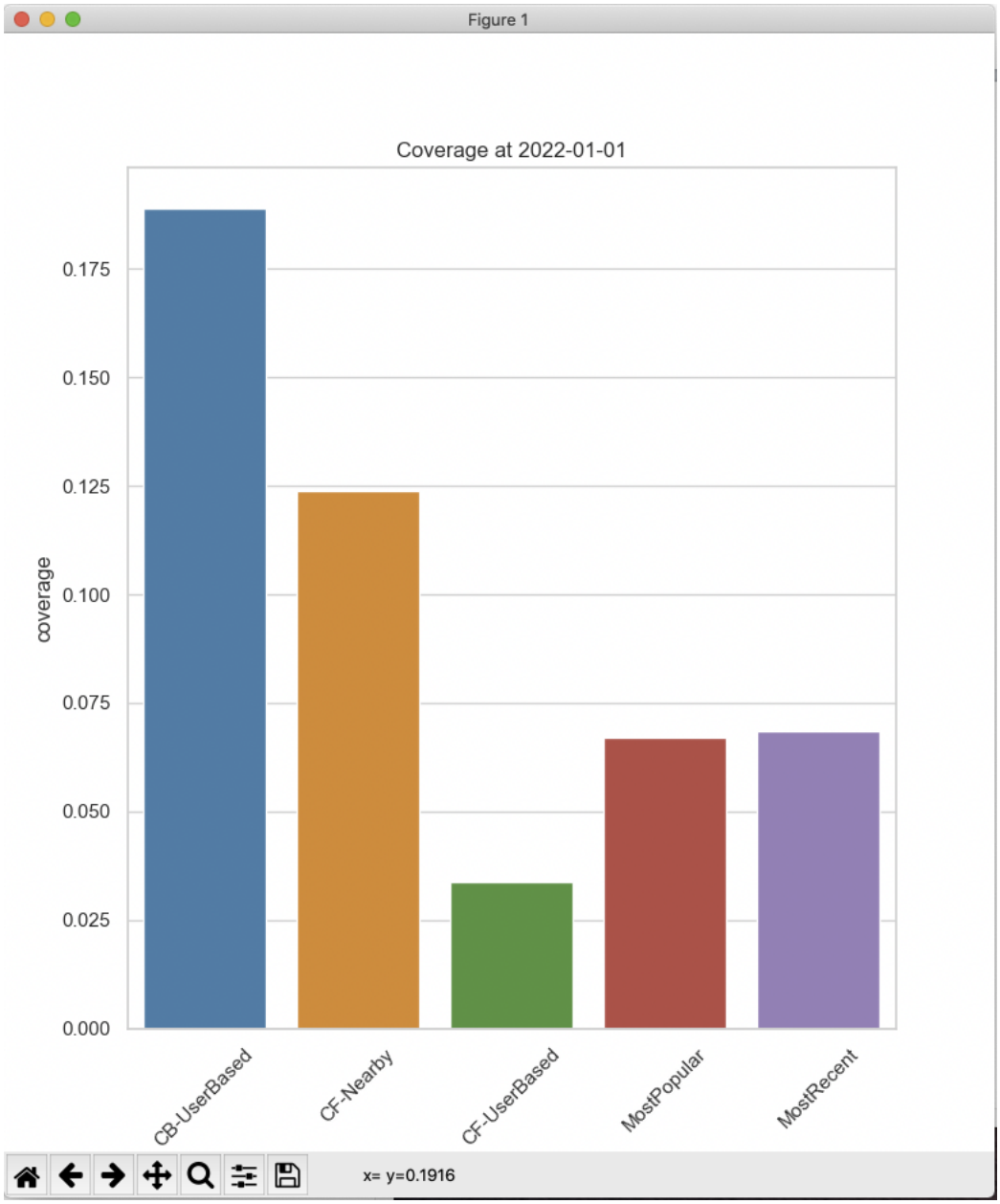

FIGURE 6.1 – Exemple de monitoring

Bibliographie

[1] adlnet/xapi-profiles.

[2] The OpenAPI Specification Explained.

[3] openapi.yaml· Noozy AI / Recommender Engine.

[4] Roussanaly AZIM et Olfa Messaoud an ASIF AHMED : Projet SmartVideo Grand Est :Knowledge about Metadata. Rapport technique, LORIA, Université de Lorraine, 06 2020.

[5] Roussanaly AZIM, Ben Ticha SONIA et Olfa MESSAOUD : Projet SmartVideo Grand Est :Spécification technique du profil xAPI pour SmartVideo. Rapport technique, LORIA, Université de Lorraine, 06 2020.

[6] Carl LAGOZE et Herbert Van de SOMPEL : The open archives initiative protocol for metadata harvesting. `http://www.openarchives.org/OAI/openarchivesprotocol.html`, 2002.

[7] Fraysse SÉBASTIEN : TRAX LRS, décembre 2021. original-date : 2019-01-31T17 :47 :46Z.

[8] Stuart WEIBEL, John KUNZE, Carl LAGOZE et Misha WOLF : Dublin core metadata for resource discovery. *Internet Engineering Task Force RFC*, 2413(222):132, 1998.

[9] Stuart L. WEIBEL et Traugott KOCH : The dublin core metadata initiative. `http://mirror.dlib.org/dlib/december00/weibel/12weibel.html#ref3`, 2000.

[10] WIKIPEDIA : *Representational state transfer*, 2020, consulté le 19 Mai 2020.