Received: 4 August 2020 | Revised: 23 December 2020 | Accepted: 22 January 2021

DOI: 10.4218/etrij.2020-0298

ORIGINAL ARTICLE

ETRI Journal WILEY

Comprehensive Knowledge Archive Network harvester improvement for efficient open-data collection and management

Dasol Kim[1] | Myeong-Seon Gil[1] | Minh Chau Nguyen[2] | Heesun Won[2] | Yang-Sae Moon[1]

[1]Department of Computer Science and Engineering, Interdisciplinary Graduate Program in Medical Bigdata Convergence, Kangwon National University, Chuncheon, Rep. of Korea

[2]CybreBrain Section, Future and Basic Technology Research Division, Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea

Correspondence
Yang-Sae Moon, Kangwon National University, Chuncheon, Rep. of Korea.
Email: ysmoon@kangwon.ac.kr

Funding information
This work was partly supported by Institute of Information & communications Technology Planning & evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2020-0-00077, Core Technology Development for Intelligently Searching and Utilizing Big Data based on DataMap) and the National Research Foundation of Korea(NRF) grant funded by the Korea government (MSIT). (No. 2019R1A2C1085311).

Abstract
With the recent increase in data disclosure, the Comprehensive Knowledge Archive Network (CKAN), which is an open-source data distribution platform, is drawing much attention. CKAN is used together with additional *extensions*, such as Datastore and Datapusher for data management and Harvest and DCAT for data collection. This study derives the problems of CKAN itself and Harvest Extension. First, CKAN causes two problems of *data inconsistency* and *storage space waste* for data deletion. Second, Harvest Extension causes three additional problems, namely *source deletion* that deletes only sources without deleting data themselves, *job stop* that cannot delete job during data collection, and *service interruption* that cannot provide service, even if data exist. Based on these observations, we propose herein an improved CKAN that provides a new deletion function solving data inconsistency and storage space waste problems. In addition, we present an improved Harvest Extension solving three problems of the legacy Harvest Extension. We verify the correctness and the usefulness of the improved CKAN and Harvest Extension functions through actual implementation and extensive experiments.

KEYWORDS
CKAN, CKAN harvester, DCAT, harvest extension, open data

1 | INTRODUCTION

Open data refer to data that anyone can easily access, (re)use, and redistribute over the Internet without particular restrictions. Representative data platforms for distributing such open data include the Comprehensive Knowledge Archive Network (CKAN) [1], Open Government Platform (OGPL) [2], and Socrata [3]. CKAN is an open-data distribution platform used in more than 40 countries, including the United Kingdom, the United States, and Canada. It is a data management project initiated by the Open Knowledge Foundation (OKF), a nonprofit foundation based on the United Kingdom. In addition to basic functions, such as data registration, publication, and statistics, CKAN supports specialized functions, such as visualization and API extraction, by combining with other open sources, including Drupal [4]. CKAN supports this specialized function under the name *Extension*. Representative extensions include Harvest for data collection, Datastore, which is an additional database, for data preview and visualization, and Datapusher for automatically uploading data to the Datastore.

This study derives the problems of CKAN itself and Harvest Extension, which we frequently meet in the practical use of CKAN and propose efficient solutions to these problems. First, we present two problems of CKAN, namely *data inconsistency* and *storage space waste*, which occur during resource deletion. Data inconsistency means information inconsistency between the metadata and the actual data store. More specifically, even though we delete the metadata in CKAN, the data store still maintains the corresponding data source files. Storage space waste is caused by an increase in data that are not accessible due to data inconsistency. CKAN has no ability to delete the actual data file store; thus, the storage space waste cannot be solved. Next, we derive three problems of Harvest Extension: *source deletion*, *job stop*, and *service interruption*. First, the source deletion problem occurs from the Harvest Extension deleting source information only from the table that stores dataset information, but not deleting much information related to the actual source. This causes a problem of unnecessarily storing data related to the deleted source. Second, the job stop problem comes from the harvesting job that cannot be stopped or deleted once executed. Job stop in the existing Harvest Extension only changes the relevant state, but it does not stop or delete the actual job; thus, its functionality is incomplete. Third, the service interruption problem is that, if the data source platform stops the service, we cannot use the data already even harvested, because existing harvesting collects only metadata and not actual data. This is a major limitation of the data distribution function.

In other commercial data platforms, including Junar [5], a manager or a team-in-charge controls these issues with the data platform. These experts are generally trying to solve the abovementioned problems, which occur in the open-source data platform and are particularly troubling platform administrators or users who have chosen CKAN. CKAN allows platform administrators to solve some problems themselves. First, the problems of inconsistency and storage waste must be addressed by the platform administrators. Next, the source deletion problem related to the Harvest Extension still limits user autonomy, and the job stop problem persists with data remaining in the queue. Lastly, the service interruption is not considered as a critical problem, because it is for data users rather than platform administrators.

This study designs and implements the actual deletion function in the data store to solve the data inconsistency and storage space waste problems of CKAN. We specifically present two scenarios of deleting resources and propose new delete functions for each scenario. Scenario 1 is when a data provider deletes resources individually. We improve CKAN, such that when the resource is deleted, we not only change its metadata to the deleted state but also remove the actual files from Filestore and Datastore if the resource has the files. Scenario 2 is when a data provider deletes an entire dataset. Currently, when deleting a dataset, CKAN deletes all resource information of the dataset from the metadata only. We improve CKAN to obtain the deleted resource information from the dataset and remove the actually stored files from Filestore and Datastore. The proposed CKAN can solve data inconsistency and storage space waste problems by applying the new delete functions for the two scenarios.

We completely redesign and implement the relevant functions to solve the three problems of Harvest Extension. First, we provide an option to delete the relevant data in addition to the existing deletion function to resolve the source deletion problem. This delete option deletes not only the source information, but also all related data from more than 20 tables related to that source. Second, we provide an additional function for deleting a job in addition to the existing job state change to solve the job stop problem. That is, for a job stop request, we improve the function to stop the job by deleting the data remaining in the queue after changing the job state. Third, we improve Harvest Extension to expand the existing method of collecting metadata only and solve the service interruption problem, such that the actual data file can be downloaded along with the metadata. This allows continuous service to existing users for data that have already been collected, even if the data source is out of service.

This study improves the CKAN utilization by solving the data inconsistency and storage space waste problems of CKAN itself and by solving the source deletion, job stop, and service interruption problems of Harvest Extension. To this end, we present in detail the design and implementation of the solution to each problem and provide it as an open source.[1] In addition, we show the operation process before and after applying each solution through actual experiments and its usefulness. The improvements herein also apply to the Smart Open Data as a Service (SODAS) project[2] of the Electronics and Telecommunications Research Institute (ETRI),[3] contributing to the expansion of international standard-based open-data platforms. We believe that the improvement presented in this paper is excellent for the enhancement of the status of CKAN as an open-source platform by enhancing the completeness of CKAN and Harvest Extension.

The rest of the paper is organized as follows: Section 2 describes the open data and CKAN as the background of

[1]The source code for the proposed CKAN and Harvest Extension is available at https://github.com/dassolkim.

[2]The international standardization-based open-data platform through the SODAS project is an improvement of the content of this paper, which can be found in (SODAS gitlab).

[3]https://www.etri.re.kr/eng/main/main.etri

TABLE 1 Comparison of open-data distribution platforms

Platform	Open source	Characteristics	Countries of use
Comprehensive Knowledge Archive Network (CKAN)	Yes	—Voluntarily participating developers around the world —Most active open-data platform —Use with other open sources for specialized features	40 countries including United States, UK, Canada, and Australia
Open Government Platform (OGPL)	Yes	—Commonly developed by US and Indian governments —Developed for transparency in public institutions	Indian Government Departments
Socrata	No	—Excellent features such as visualization and analysis	US states governments

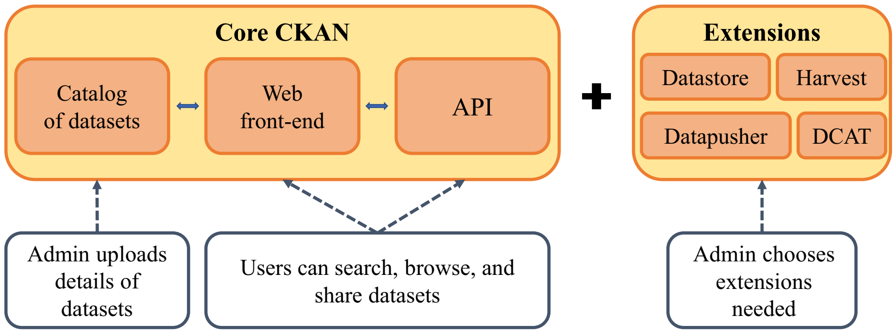

Core CKAN
Catalog of datasets
Web front-end
API
Extensions
Datastore
Harvest
Datapusher
DCAT
Admin uploads details of datasets
Users can search, browse, and share datasets
Admin chooses extensions needed

FIGURE 1 Operation structure of Comprehensive Knowledge Archive Network (CKAN) and major extensions

this paper; Section 3 analyzes the operations of CKAN and Harvest Extension and explains their problems; Section 4 designs improvements to address those problems and describes the implementation details; Section 5 confirms, by experiment, that the solutions proposed in Section 4 work correctly; and Section 6 finally summarizes and concludes the paper.

2 | BACKGROUND AND RELATED WORK

2.1 | Open data and distribution platforms

A platform for deploying, managing, and sharing open data has attracted much attention with the recent increase in data disclosure. Open data refer to data that can be freely used, reused, and redistributed by everyone. It is mainly the disclosure of government public data for transparency, releasing social and commerce value, and partitioning and engagement [6]. As such, the focus is mainly on public data disclosure. The use of open data is gradually increasing depending on the will of governments. We call the environment for publishing and distributing such open data the *open-data distribution platform*. Among several open-data distribution platforms [7], the most widely used is the open-source project "CKAN." In addition to CKAN, OGPL is used as an open source, whereas Socrata and Junar are used as commercial platforms.

Table 1 compares the characteristics of CKAN and other platforms and summarizes the countries and institutions currently in use. The most actively developed CKAN is used as a government data portal in more than 40 countries, including the United States, the United Kingdom, Canada, and Australia. OGPL is an open-data platform developed by the United States and Indian governments currently used by Indian Government Departments. Socrata is an open-data platform developed in the United States and currently used as a data portal by the US federal government. The US government built a public data portal as Socrata in 2009, but it switched to CKAN in 2013 and is currently operating. Similar to the United States, Australia's data portal also initially used other platforms, but now switched to CKAN.

CKAN [1,8] is an open-source data management solution, which is not a file repository, but a data distribution platform [9]. CKAN is widely used because of four main reasons. First, it is an open-source platform. Second, it has easy features for data management. Third, various support is available in connection with the community, and many developers are involved. Fourth, it supports various functions for data providers and users. Therefore, CKAN has become the most popular open-data distribution platform widely adopted by many governments, including the United States (data.gov), United Kingdom (data.gov.uk), Canada (open.canada.ca), Australia (data.gov.au), and the European Union (europeandataportal.eu) [10].

2.2 | CKAN and CKAN extensions

CKAN uses PostgreSQL [11] for database, SQLAlchemy [12] for object-relational mapping (ORM) [13], and Apache Solr [14] for search engine. CKAN provides extensions to expand functionality, thereby allowing users to extend CKAN as needed. Figure 1 shows the operation structure of CKAN

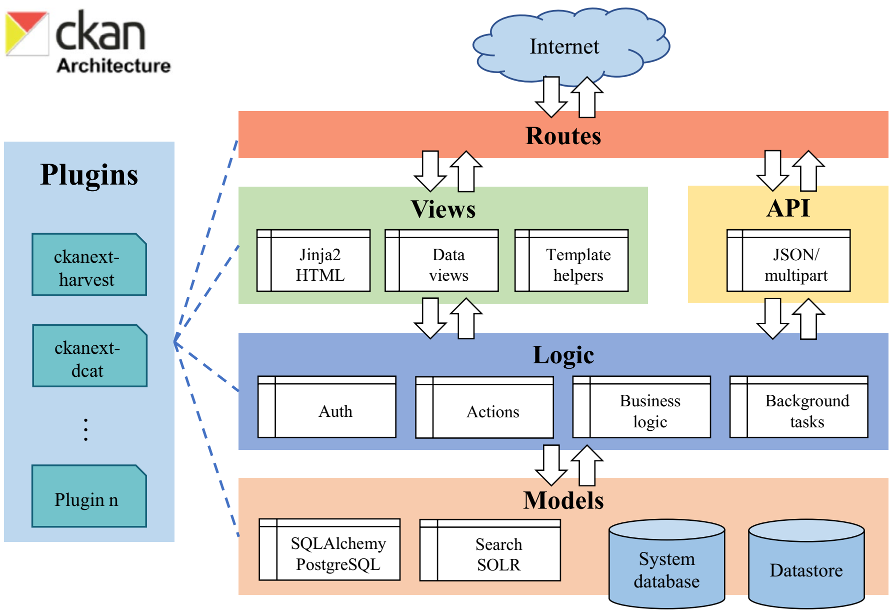


FIGURE 2 Internal source code structure of Comprehensive Knowledge Archive Network (CKAN)

and major extensions based on the user type. In CKAN, users are divided into data providers and data users. Data providers are primarily national and local governments releasing data directly through web interfaces and APIs or collecting and disclosing data from the outside. Collecting data from outside is called harvesting. Data users search for the desired data using the web interface and import the data from the CKAN server through the API. At this time, the user can search for the open dataset through the basic classification system provided by CKAN. The basic classification system of CKAN includes Organization, Group, Tag, Format, and License [15].

Figure 2 shows the internal source code structure of CKAN. Briefly describing each module, Routes define a connection between the CKAN URL and Views that handle requests and provide responses. Views use the Action function to read and update data to handle requests and render the Jinja2 [16] template to return responses. In Views, the Template Helper is used for the code frequently reused or too complex to be included in the template itself. Logic contains the most important functions, such as Actions and Auth, which are responsible for the internal operation of the platform. The Action's functions are provided to the outside with API URLs of the same names. Models are responsible for storing and retrieving objects in the PostgreSQL database through ORM SQLAlchemy. CKAN also uses Solr, a search engine, to exploit the customized schema considering specific search needs. In this way, the inside of CKAN is composed of four layers serving as a data distribution platform. The plug-ins on the left side of the figure represent extensions for additional functions of CKAN. In addition, CKAN provides a basic template such that developers can directly create a new extension if there is no desired extension.

CKAN has more than 230 functions registered as extensions, including visualization, document preview, custom theme, other storage, site link, and metadata management. Representative examples include the DCAT Extension for linked data [17,18], CKAN harvester or DCAT [19] RDF/JSON harvester for data collection, and DB Extension for storage linkage to MongoDB [20] or Amazon S3 [21]. We analyzed herein the following extensions related to harvesting.

- Datastore is an additional data store that stores the actual resource files in database tables.
- Datapusher automatically uploads data to the Datastore.
- Harvest performs data collection between the CKAN platforms.
- DCAT provides the DCAT standard plug-in for use with CKAN.

2.3 | Related work

Recent research on CKAN and open-data platforms include those of Millette et al. [22] and Elmekki et al. [23] analyzed the functions of open-data platforms from the perspective of data users and increased the open-data usability. These studies presented novel solutions for users to more efficiently use datasets registered in existing open-data platforms. Macedo [24] proposed the CKAN Extension for easier data publication and management compared with the existing CKAN. Varon-Capera et al. [25] proposed a visualization tool that provides visual analysis to increase the CKAN data utilization. Scholz et al. [26] proposed a CKAN plug-in that stores actual data in HDFS by utilizing CKAN as a metadata catalog.

We perform herein a CKAN function analysis from the perspective of data providers and users and platform administrators. Unlike that in [22,23], our analysis focuses on the most widely used CKAN among open-data platforms. Compared

TABLE 2 Major functions of Comprehensive Knowledge Archive Network (CKAN)

Function	Explanation
Dataset management	Creation, modification, and deletion of datasets
Resource management	Creation, modification, and deletion of resources
User management	Creation, modification, and deletion of users
Activity stream	Change management of datasets, resources, and users
Keyword auto complete	Autocomplete function of words
Translation	Translation function of words
Tagging	Tagging function for datasets
Grouping	Group management of datasets
Follower	Following function for users, datasets, organizations, and groups
Linked data	Converting metadata of dataset stored in DB to RDF
Rest API	Function access through Rest API
Authority management	Access controls for each user
Search	Searching function for datasets
Harvesting	Collection of datasets of CKAN instances
Social	Sharing of dataset information through Facebook and Twitter
Data preview	Web preview functions for structured data
Geospatial	Geospatial information for datasets presented on the map
License management	License settings for each dataset
Extension	Function extensions to meet user needs
Metadata customization	Use of new metadata in addition to the basic metadata
Permanent URLs	Grant a permanent URL to each dataset

with [24,25], we propose various solutions in the viewpoint of data providers and platform administrators. In addition, our method fundamentally solves the data inconsistency and storage waste problems compared with [26]. In summary, we present a novel solution for each platform administrator and data provider and user to use harvesting efficiently.

3 | FUNCTIONAL ANALYSIS OF CKAN HARVESTING

This section analyzes the CKAN behavior, focusing on the data collection, which is the most important function of the open-data platform. Section 3.1 analyzes the CKAN functions based on the user role of the open-data platform. Section 3.2 analyzes the functions of essential harvesting extensions.

3.1 | CKAN function analysis based on user roles

Table 2 summarizes the major functions of CKAN. It provides various functions from the dataset, resource, and user management basically provided by the data distribution platform to the metadata management of each dataset. Open-data platform users can be divided into data providers, data users, and platform administrators based on their roles. The CKAN functions according to these user roles are summarized as follows:

- Data providers: dataset, resource, and organization management.
- Data users: faceted search, data preview, and personalization.
- Platform administrators: federation and customization.

The advantage of using CKAN as a data provider is that various functions for data disclosure and management are available through the web interface. The data provider can manage metadata for various datasets and resources at his/her convenience. The metadata that should be basically registered together with a dataset include the dataset name, organization, and description. The provider may also select representative tags for the dataset, license information of the open data, and whether to disclose the dataset. Other additional inputs include provider information, administrator information for datasets, and custom fields for entering supplementary information in a key/value format.

By registering a package that is the metadata for a dataset, we can register the resources that store the actual data for that dataset. Resources can be registered in two ways: (a) provide a URL such that we can access the actual file on the Internet and (b) upload actual data files and store them in the file storage. If we upload an actual resource file directly, its name and format are automatically registered, and we can change that information or add other descriptions. Please refer to Appendix A for an ER diagram of the tables that store relevant metadata when creating a dataset.

Next, we analyze the CKAN functions from the data user perspective. For data users, the biggest advantage of CKAN is the easy use of searching and exploring huge data. CKAN provides a "faceted search" technology, such that users can gradually narrow the search scope by applying taxonomic filters, such as tags and formats. It also provides data preview in a form familiar to the user through tables, graphs, and maps. These data previews and visualizations can be extended through extensions; hence, many data portals currently provide various visualization and preview functions. Please refer to Appendix B for an example screenshot of a Faceted Search [27].

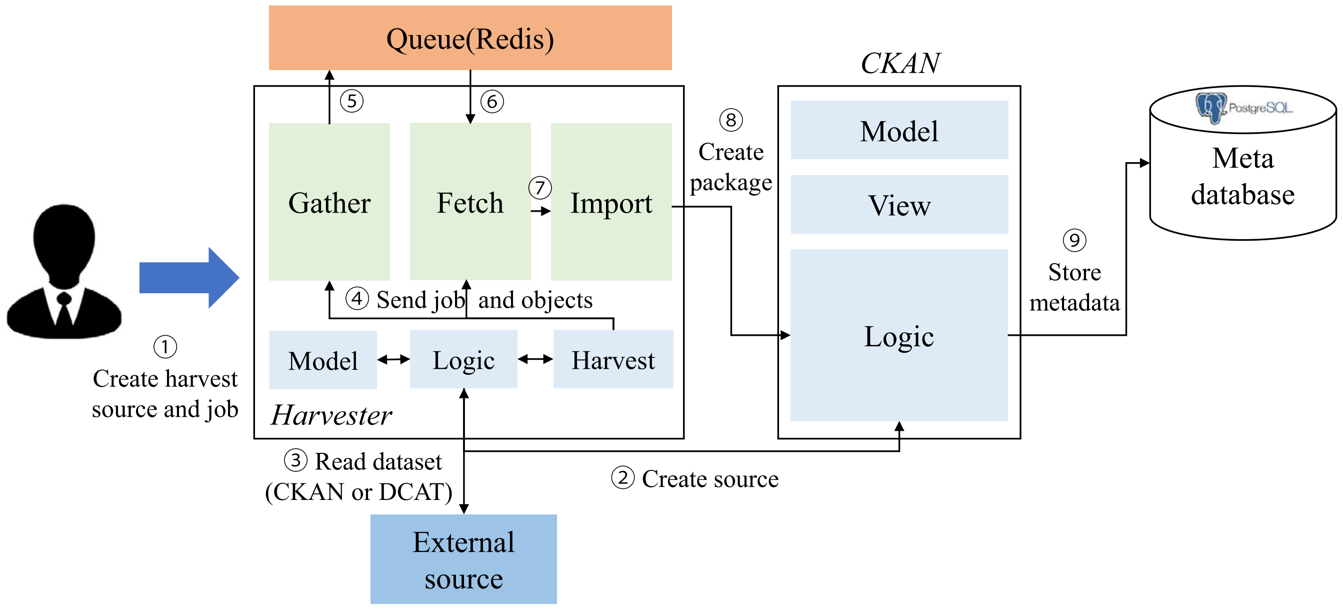


FIGURE 3 Operation steps of the Comprehensive Knowledge Archive Network (CKAN) harvester

Finally, from the platform administrator's point of view, the advantage of using CKAN is that it is easy to integrate and customize with other platforms. Using CKAN, we can connect with other CKAN networks and scale up the data portal through data sharing. The simplest way to connect with other data portals using CKAN is to collect and share open data through data harvesting. We describe the data harvesting issue in detail in Section 3.2.

3.2 | Essential extensions for data harvesting

This section presents in detail the extensions related to the data harvesting function. We first describe Datastore and Datapusher, which are the most widely used extensions. We next explain Harvest Extension.

Datastore is an additional DB, and unlike the System Database that stores metadata, it stores actual data as tables. Datastore stores structured data and supports CSV and Excel formats in CKAN. We used Datastore because it can provide data preview in the portal UI by storing actual data and support additional functions, such as data search, filtering, and modification through the API of Datastore itself. Datapusher is an extension for data loading, which automatically loads the data into Datastore if the resource data are in CSV or Excel format. In other words, Datapusher is an Extension for automating Datastore.

We basically use Harvest Extension for data collection in CKAN and DCAT extension for interconnection with the DCAT standard. We can also use the DCAT RDF harvester by installing Harvest Extension and adding DCAT Extension. The CKAN harvester supports Redis [28] and RabbitMQ [29] as the queue components. We currently mainly use Redis. Please refer to Appendix C for an ER diagram of the tables to be generated when Harvest Extension is added.

Figure 3 shows the CKAN harvesting process. Each step works as follows: ① The user creates a source to perform harvesting and requests job execution; ② the harvester stores source information generated through API in DB inside CKAN; ③ when the job runs, it checks the source URL and retrieves dataset information from the source; ④ the harvester creates an object by mapping one to one with the retrieved dataset information; ⑤ it pushes the generated job and object IDs to the queue; ⑥ when the object ID is popped from the queue, it checks whether the object is stored in DB and saves the timestamp; and ⑦ the object moves to the Import phase after the verification. ⑧ In the Import phase, the harvester requests CKAN to create and update the dataset. ⑨ CKAN stores or updates the dataset and the related information in DB inside CKAN. In Step 3, the basic CKAN and DCAT RDF harvesters operate differently. The CKAN harvester uses the package_search API to read the dataset information on the same CKAN platform. On the contrary, the DCAT RDF harvester requests information by attaching RDF Endpoint [30] and receives the RDF files returned upon request. Subsequently, DCAT Extension performs the mapping for processing in CKAN and reads the dataset information.

4 | IMPROVED CKAN AND HARVESTING EXTENSIONS

In this section, we analyze the problems of the existing CKAN and Harvest Extension and present solutions for these problems.

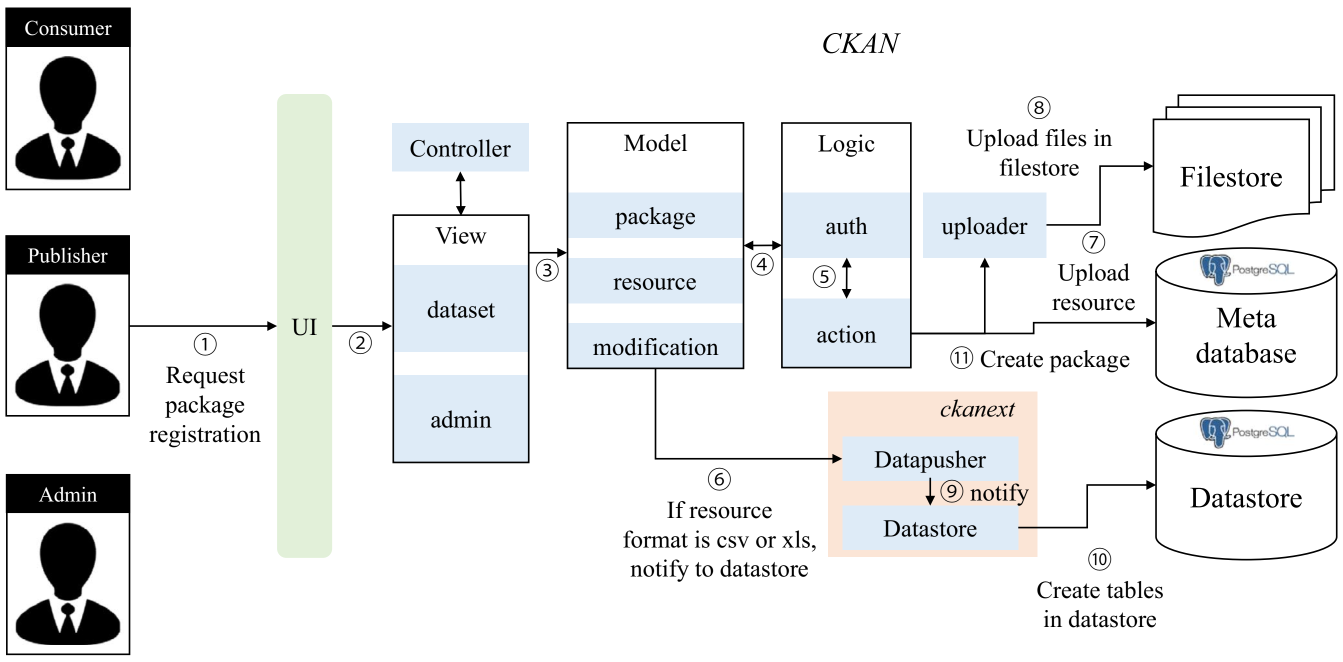


FIGURE 4 Data registration process in Comprehensive Knowledge Archive Network (CKAN) through Datastore

4.1 | Problem analysis of existing CKAN

We analyze herein the data registration and deletion processes of the existing CKAN and present the problems caused by these processes. Figure 4 shows the data registration process of CKAN. Each step is described as follows: ① The data provider creates a package (ie, metadata) through the UI; ② CKAN delivers information registered by the provider to View through the API; ③ it creates a model to store in the actual DB using Logic API; ④ CKAN provides the work that creates the actual dataset by referring to the model; ⑤ it checks the authentication and authorization of the user to create the dataset; ⑥ after registering the package, CKAN delivers the resource format to Datapusher if the resource is registered; ⑦ it stores the metadata of the resource and requests the actual files to be stored in Filestore; ⑧ it stores the actual files in Filestore; ⑨ Datapusher uses its own API to deliver information to Datastore; ⑩ CKAN processes the resource received from Datastore internally and creates as a table; and ⑪ it completes the dataset and resource creation by storing the metadata. Through these steps, the metadata are stored in the meta-DB. The resources are basically stored in Filestore, and the resources in CSV/Excel format are separately stored in Datastore.

Figure 5 shows the data deletion process, with each step described as follows: ① The data provider requests the dataset deletion; ② CKAN delivers the dataset information to be deleted to View through the API; ③ it retrieves the dataset information from the meta-DB and delivers it to Model; ④ it checks the dataset information and delivers an actual deletion request to Logic; ⑤ it checks the authentication and authorization of the user to delete the dataset; ⑥ CKAN changes the status information of the data to be deleted from “active” to “deleted”; ⑦ the deleted dataset is moved to the trash bin on the platform administrator page; ⑧ the platform administrator finally decides whether to completely delete it from the meta DB; ⑨ the administrator executes a command to empty the trash; and ⑩ CKAN deletes all information related to the deleted dataset, such as tags and resources, from the meta-DB. As shown in the mentioned steps, the dataset deletion is performed sequentially. Ultimately, the platform administrator should perform the purge command on the administrator's page to completely delete all relevant information from the meta-DB. At this time, the metadata related to the resources belonging to the deleted dataset are also deleted.

The CKAN dataset deletion process described so far causes two big problems. The first problem is *data inconsistency*, which means a mismatch between the meta-DB and the actual data storage. The second problem is *storage space waste*, which means that inaccessible data are unnecessarily maintained due to the data inconsistency. Table 3 summarizes these two problems.

Figure 6 shows the data inconsistency problem in CKAN. The existing CKAN incurs the data inconsistency problem because it deletes the metadata only without deleting the resource files from the actual storage. In the figure, (A) shows the data inconsistency between meta-DB and Filestore, and (B) shows the data inconsistency between meta-DB and Datastore.

This data inconsistency causes serious problems in managing data as the platform becomes larger. In addition, archiving the original data files of resources that are no longer

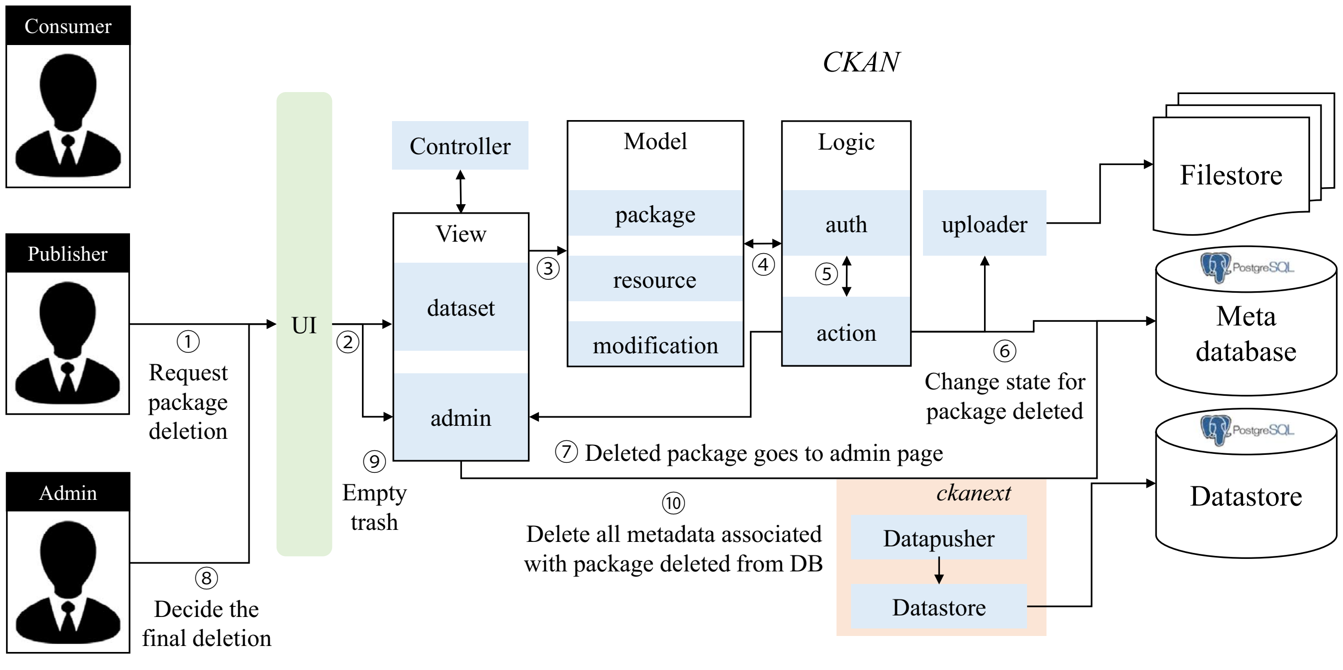

Consumer
Publisher
Admin
CKAN
UI
Controller
View
dataset
admin
Model
package
resource
modification
Logic
auth
action
uploader
Filestore
Meta database
Datastore
ckanext
Datapusher
Datastore
① Request package deletion
②
③
④
⑤
⑥ Change state for package deleted
⑦ Deleted package goes to admin page
⑧ Decide the final deletion
⑨ Empty trash
⑩ Delete all metadata associated with package deleted from DB

FIGURE 5 Data deletion process in Comprehensive Knowledge Archive Network (CKAN) through Datastore

TABLE 3 Comprehensive Knowledge Archive Network (CKAN) problems caused by the deletion process

Problem	Problem details
Data inconsistency	—Data inconsistency occurs in between meta-DB and Filestore —Data inconsistency occurs in between meta-DB and Datastore
Storage space waste	—Storage space is wasted in Filestore —Storage space is wasted in Datastore

distributed causes serious storage space waste. Storage space waste arises from the data inconsistency and the actual data storage having no delete function. In other words, the existing CKAN does not support the delete function to remove data files stored in Filestore and Datastore. These are pointed out by many CKAN users as big problems. Major national portals deal with these problems in their own way, but no common solutions have yet been provided.

4.2 | Problem analysis of existing CKAN harvesting

The open-data platform basically performs one harvesting operation at a time. It queues other operations if the previous one is not completed and executes them later. Looking at the harvesting steps based on the queue are as follows. First, the Gather step operates in a separate queue, gather_consumer, and receives job_id as input. Second, the Fetch step receives and processes object_id through fetch_consumer. Third, the Import step performs the operation through the CKAN internal API without going through the queue. Please refer to Appendix D for the details of the internal operations performed in the harvesting steps.

We identified three major problems in Table 4 while analyzing the CKAN harvesting function. First, the *source deletion* problem occurs when deleting a source from the Harvest Extension because it only changes the source table to an inactive state, but does not delete the relevant data. Figure 7 shows a comparison of the example contents of the package table of storing dataset information and the harvest_source table of storing source information after deleting the source. After deletion, only two tables are marked as inactive. The data related to the source are not deleted.

Second, the *job stop* problem is that once a job is executed, there is no way to stop or delete it in the middle. According to the source code analysis, we confirmed that the current job stop function only changes the status in DB, but does not stop the job. In CKAN, the harvesting job runs in the background; thus, the job to be stopped does not actually stop and continues to run, causing system resource waste and waiting for other jobs.

Third, the *service interruption* problem occurs when the remote source platform that provides the harvesting service no longer supports the service. In this case, the platform that collects and distributes the data can no longer provide data to the user because the existing harvesting collects only the metadata, and the actual data exist in the external sources. On the contrary, users with access to metadata may want to receive continuous data services, even if limited; however, the current Harvest Extension does not support this service.

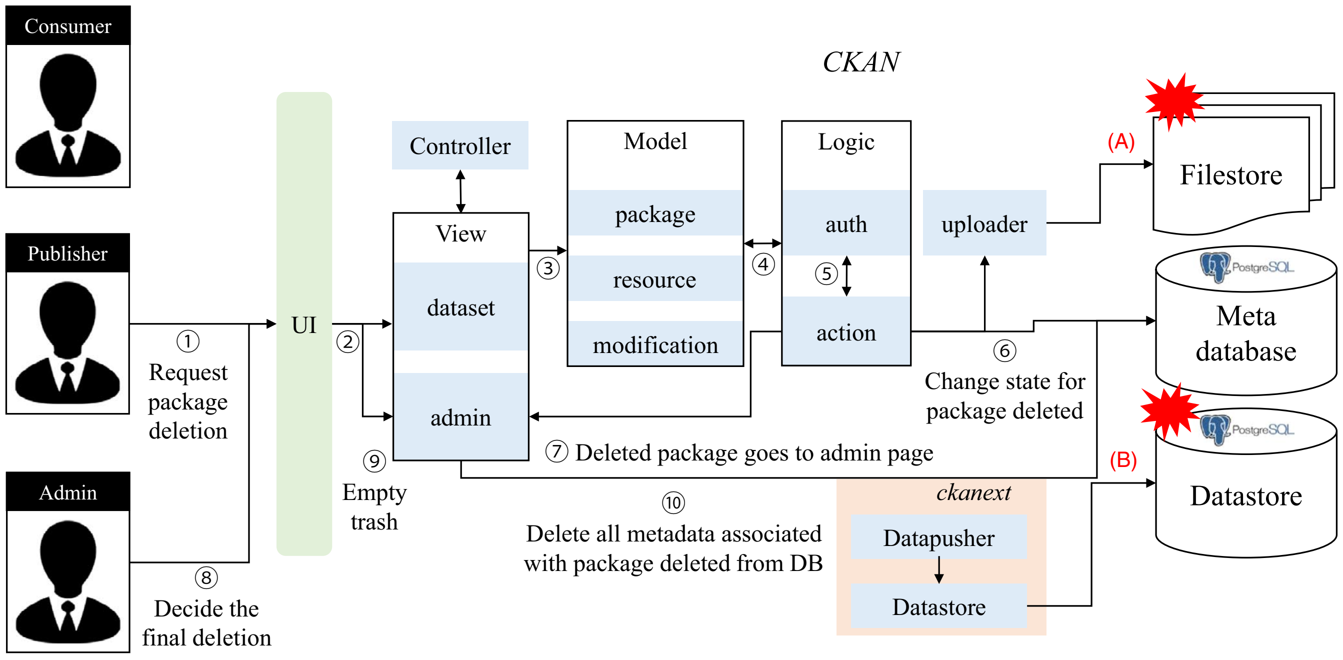


FIGURE 6 Data inconsistency problems caused by the deletion process

TABLE 4 Three major problems of the existing Harvest Extension

Problem	Resulting phenomenon
Source deletion	—Waste of DB storage space —User autonomy restriction
Job stop	—Waste of system resources —Other job waiting situations
Service interruption	—Incomplete data distribution

4.3 | Design and implementation of the improved CKAN

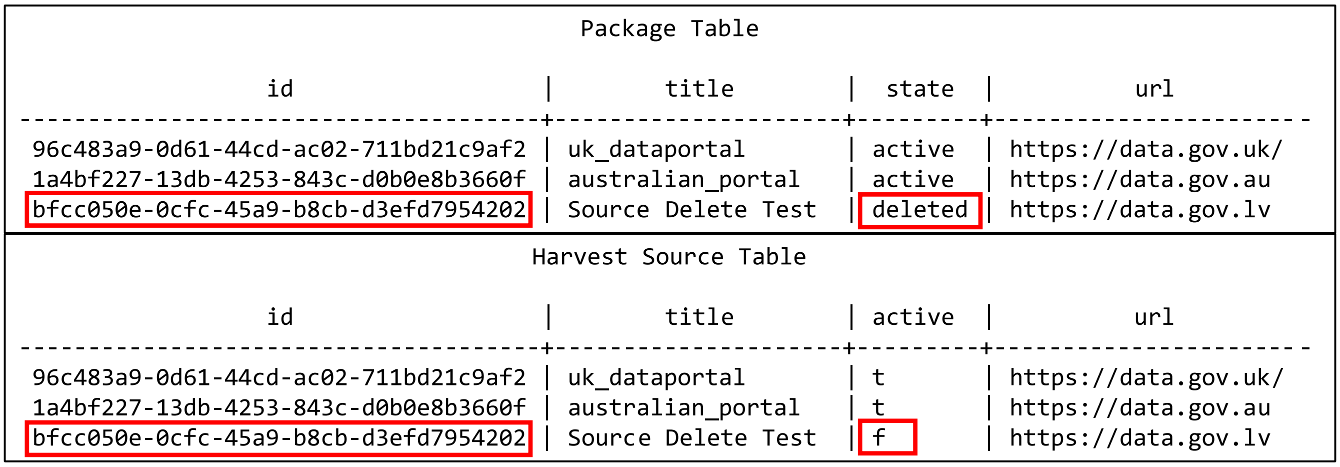
In this section, we design and implement a practical solution to the data inconsistency and storage space waste problems of CKAN. To this end, we first present two scenarios in which the resource is deleted. Based on these, we then explain how to solve the problems. Scenario 1 is the case where a data provider deletes a resource registered by itself. Scenario 2 is the case where a platform administrator finally deletes a dataset. In Scenario 1, when a data provider deletes a resource, the existing CKAN changes its resource status to "deleted" in the metadata and no longer distributes the data. To solve these two problems, we improve CKAN to change the resource status in the metadata and at the same time delete the resource files in Filestore and Datastore. Figure 8 illustrates the process of solving the resource deletion problems in Scenario 1. As shown in the figure, if the provider requests deletion (①–②) and the resource status changes in metadata (③–⑥), the improved CKAN also deletes the original data files of each resource from the storage (⑦–⑧).

In Scenario 2, the data provider requests the dataset deletion, and the platform administrator finally performs the deletion function. Even in this case, the existing CKAN still maintains the actual data files of the deleted resource in each storage. To solve this, when the platform administrator performs the purge function, we improve CKAN to perform the actual delete function after checking whether the data files are stored. Figure 9 shows the improved process of solving the resource deletion problem in Scenario 2. When the data provider requests to delete the dataset (①–②), it is deleted from the metadata (③–⑥) and put into the trash bin of the platform administrator (⑦). When the platform administrator requests the delete function from the trash bin (⑧), it checks whether or not the data files are stored in the data storage. If the original data files remain in Filestore or Datastore, it performs the actual deletion in each storage and metadata (⑩–⑫).

The existing CKAN has no actual deletion function of data storage for the two abovementioned scenarios described. We implement herein the deletion function for the actual storage and integrate it into the existing CKAN function. We implement file_remove, which is a function that deletes files stored in Filestore, and dsfile_remove, which is a function that deletes tables stored in Datastore. The readers are referred to the source code (https://github.com/dassolkim/default-ckan) for details of the implementation.

4.4 | Design and implementation of the improved harvesting extensions

In this section, we design and implement solutions to the three problems of Harvest Extension. The first is the source



```
                          Package Table

                  id                  |       title        |  state   |          url
--------------------------------------+--------------------+----------+-----------------------
 96c483a9-0d61-44cd-ac02-711bd21c9af2 | uk_dataportal      | active   | https://data.gov.uk/
 1a4bf227-13db-4253-843c-d0b0e8b3660f | australian_portal  | active   | https://data.gov.au
 bfcc050e-0cfc-45a9-b8cb-d3efd7954202 | Source Delete Test | deleted  | https://data.gov.lv
                       Harvest Source Table

                  id                  |       title        | active   |          url
--------------------------------------+--------------------+----------+-----------------------
 96c483a9-0d61-44cd-ac02-711bd21c9af2 | uk_dataportal      | t        | https://data.gov.uk/
 1a4bf227-13db-4253-843c-d0b0e8b3660f | australian_portal  | t        | https://data.gov.au
 bfcc050e-0cfc-45a9-b8cb-d3efd7954202 | Source Delete Test | f        | https://data.gov.lv
```

FIGURE 7 Change in tables after the source deletion in the Comprehensive Knowledge Archive Network (CKAN) harvester

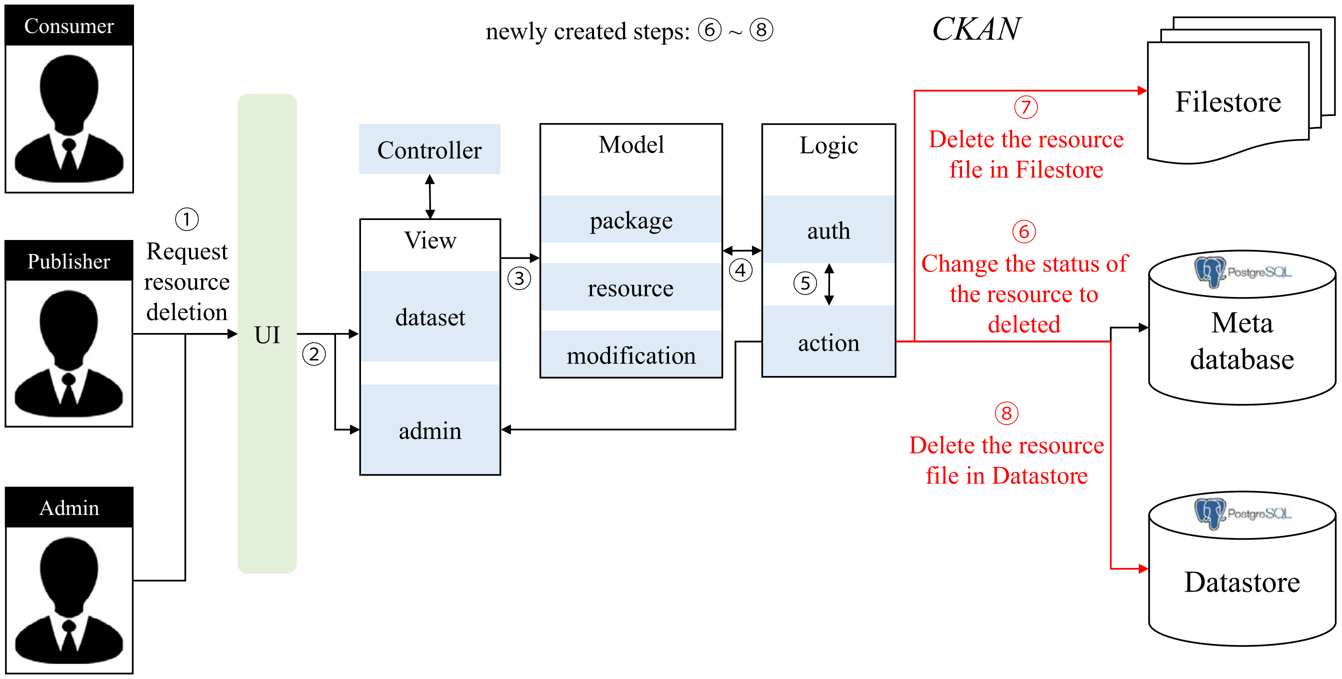

Consumer
Publisher
Admin
newly created steps: ⑥ ~ ⑧
CKAN
① Request resource deletion
UI
②
Controller
View
dataset
admin
③
Model
package
resource
modification
④
Logic
auth
⑤
action
⑦ Delete the resource file in Filestore
⑥ Change the status of the resource to deleted
⑧ Delete the resource file in Datastore
Filestore
PostgreSQL
Meta database
PostgreSQL
Datastore

FIGURE 8 Scenario 1: problems and solutions when deleting resources

deletion. In the existing extension, deleting the source changes only the source table status while keeping the related data. Accordingly, we improve the source deletion function of Harvest Extension to solve this problem. In the improved deletion function, when the user deletes the source, it deletes data from 20 related tables together and finally deletes the source information. We can solve the source deletion problem using this function, consequently saving the DB storage space. In Harvest Extension 1.1.0 [31], a function similar to the deletion function in this study was included as a "delete and clear source." The two functions are similar in terms of deleting the source and related data together. However, a difference is that while version 1.0.0 does not delete the actual source information, our deletion function completely deletes it, even the source information. We designed and implemented it to be able to select the "deleted clear source" of version 1.0.0 and the "delete complete source" of the proposed solution as options.

Second, the job stop problem comes from the fact that no actual function exists for stopping jobs in the existing Harvest Extension. To solve this problem, we developed a function to actually stop and delete the job. Figure 10 shows the operation process of the job stop function designed in this study. First, when the user requests a job stop operation, it calls the job_abort function to change the DB status (①–③). Next, it checks the information in the Gather and Fetch queues and

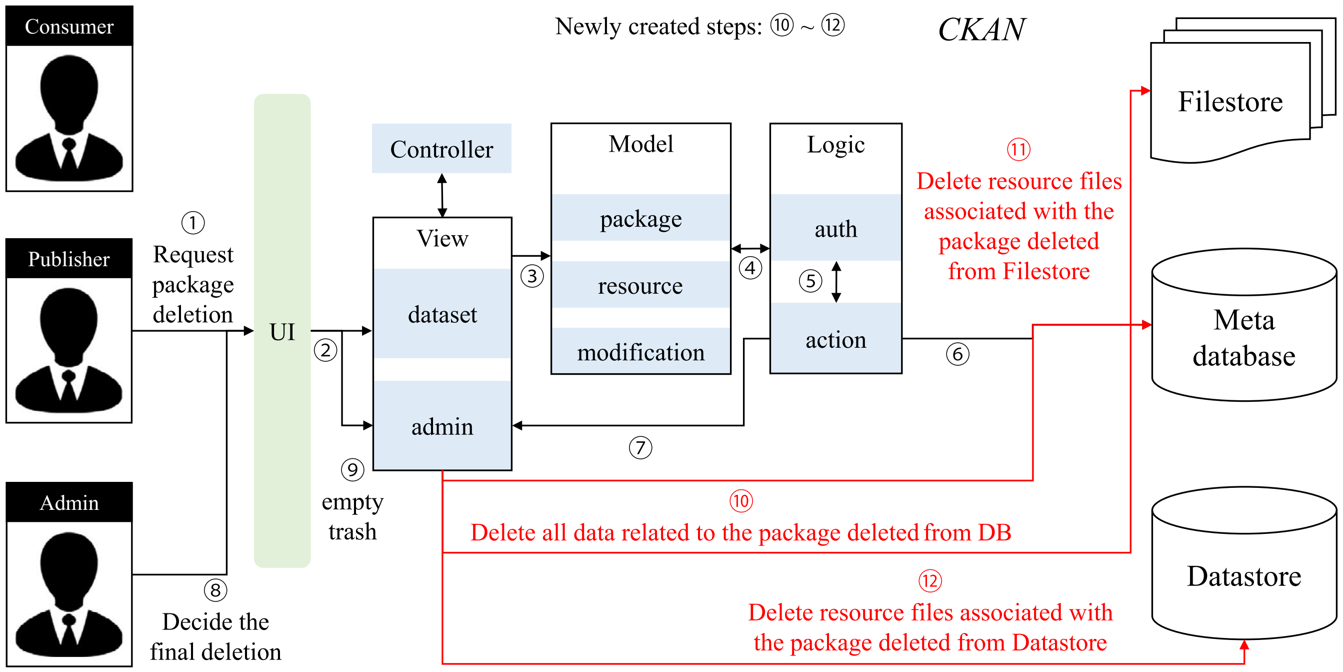

Newly created steps: ⑩ ~ ⑫
CKAN
Consumer
Publisher
Admin
① Request package deletion
UI
Controller
View
dataset
admin
Model
package
resource
modification
Logic
auth
action
Filestore
Meta database
Datastore
②
③
④
⑤
⑥
⑦
⑧ Decide the final deletion
⑨ empty trash
⑩ Delete all data related to the package deleted from DB
⑪ Delete resource files associated with the package deleted from Filestore
⑫ Delete resource files associated with the package deleted from Datastore

FIGURE 9 Scenario 2: problems and solutions when deleting datasets

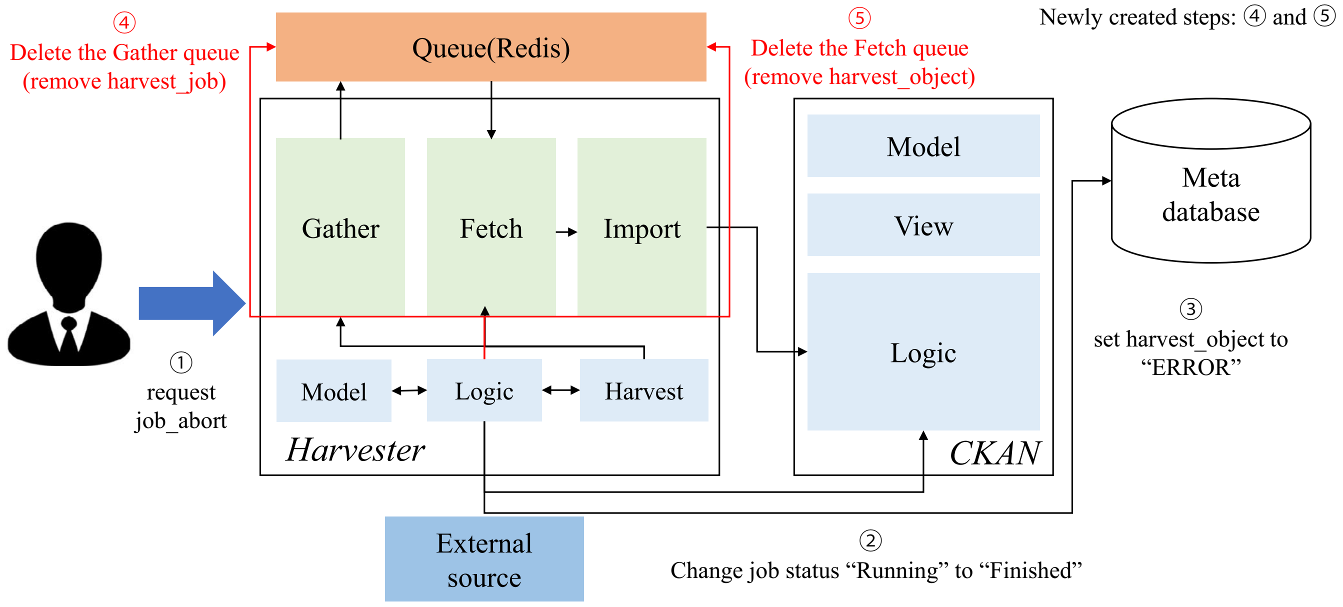

④ Delete the Gather queue (remove harvest_job)
Queue(Redis)
⑤ Delete the Fetch queue (remove harvest_object)
Newly created steps: ④ and ⑤
Gather
Fetch
Import
Model
Logic
Harvest
Harvester
Model
View
Logic
CKAN
Meta database
③ set harvest_object to "ERROR"
① request job_abort
External source
② Change job status "Running" to "Finished"

FIGURE 10 Redesign of Harvest Extension for the improved job stop

deletes the requested job and the objects created from the job (④–⑤). After clearing the queues, it returns the number of deleted objects and finishes the job stop function. Through this process, the DB status change and queue data deletion work together, resulting in successful job stop and deletion.

Third, the service interruption problem means that it can no longer provide a dataset already harvested by CKAN. We solve this problem by improving Harvest Extension to retrieve and store the actual data files that can be collected together with the metadata when harvesting. Figure 11 shows the entire harvesting process of collecting not only the metadata, but also the actual data files. As shown in the figure, the improved Harvest Extension stores the actual data files in separate data stores, Filestore and Datastore. We design this function in conjunction with the Datapusher Extension, where it stores CSV/Excel format files in Datastore as tables and other format files or large files in Filestore as regular files. In the figure, the parts of ⑩–⑬ are the additionally implemented function. We implemented it by extending the harvesting function and integrating it with the existing collection module.

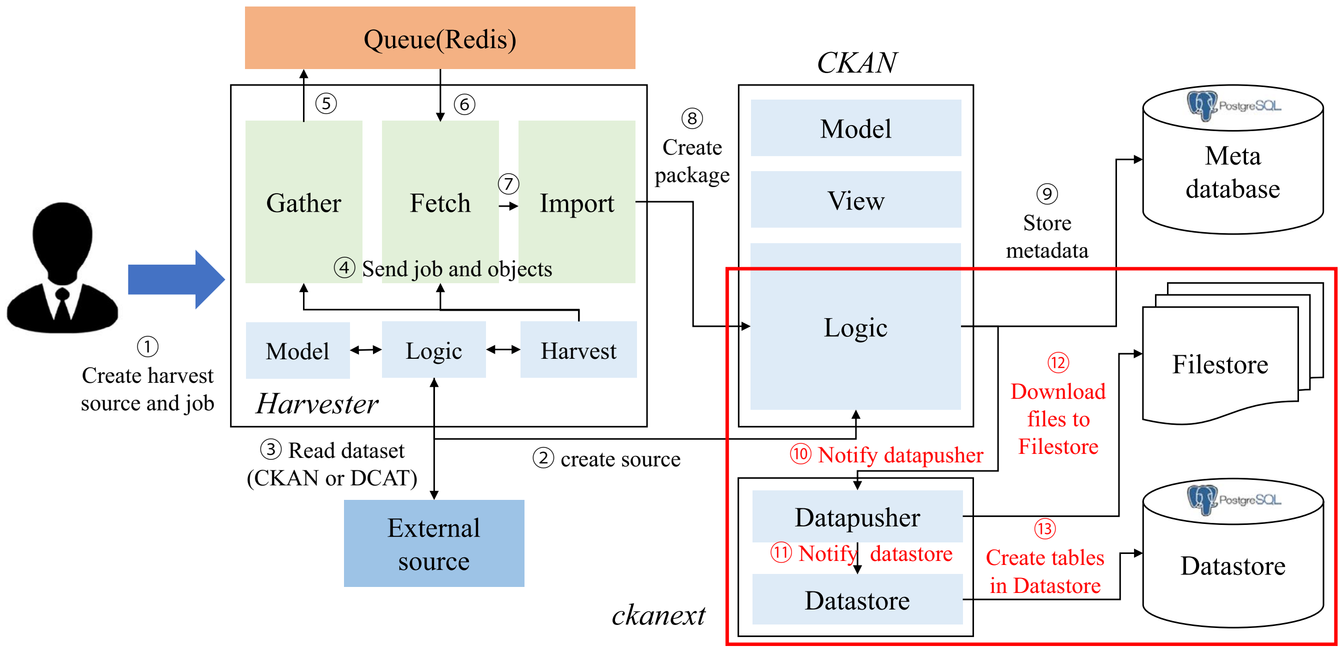


FIGURE 11 Improved harvesting operation for solving the service interruption problem

TABLE 5 Version information of the projects used for development.

Project	Version	Project	Version
Comprehensive Knowledge Archive Network	2.6.2	Harvest Extension	1.1.0
Datapusher extension	0.0.12	DCAT extension	0.0.5

5 | EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate the changes in the CKAN and the extension improved in this study. We used Python as the development language in the same way as CKAN and PostgreSQL as the DB. We ran CKAN and the extensions used for the implementation on the Linux server of Ubuntu 16.04.5 LTS. Table 5 presents the version information of the projects used for development.

5.1 | Evaluation of the improved CKAN

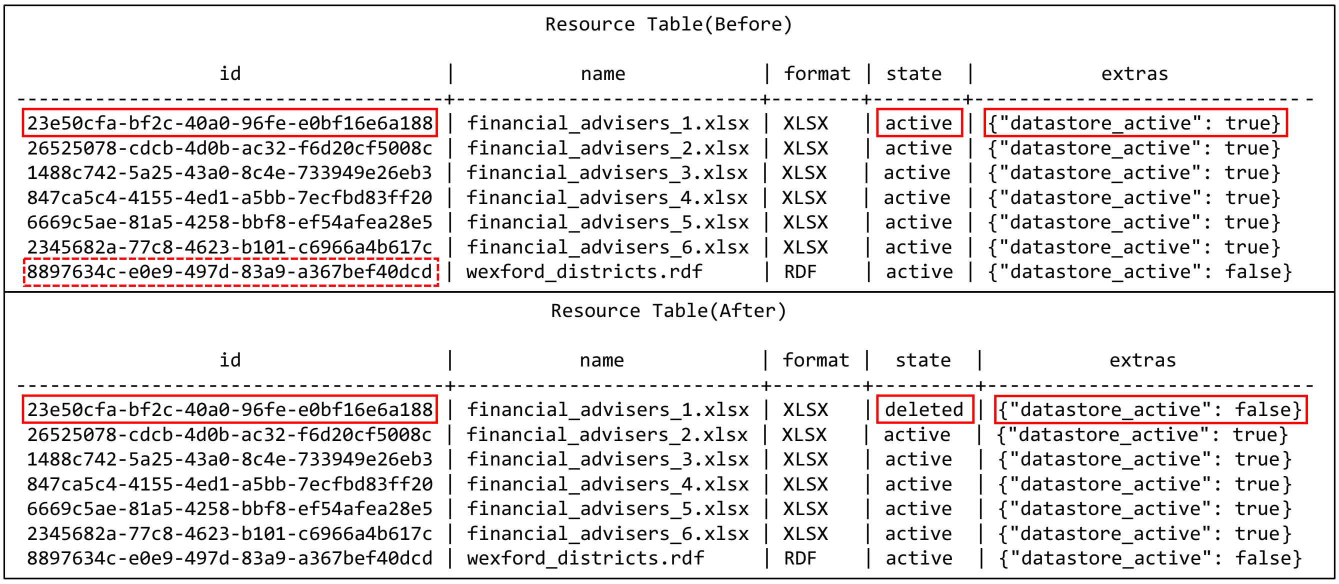
We confirm herein that the improved CKAN correctly solves the data inconsistency and storage space waste problems through actual experiments. To do this, we experiment with the delete function of actual data stores for both scenarios. Scenario 1 is a case of deleting resources individually, whereas Scenario 2 is a case of deleting an entire dataset, not an individual resource.

The evaluation result for Scenario 1 verified that when a data provider deletes a resource, its corresponding files in Filestore and Datastore are correctly deleted. For the experiment, we created a dataset with seven resources. In Figure 12, the top six resources are those stored in Datastore, whereas the bottom dotted resource is that stored in Filestore. Figure 12 shows the resource table before and after deleting a resource marked as solid boxes. The figure shows that the status of the solid-line resource changes from "active" to "deleted." "datastore_active" also changes from "true" to "false." Figure 13 compares the top 10 tables before and after the resource deletion to determine whether this deleted resource is actually deleted from Datastore. Figure 12 shows that the resource corresponding to the solid boxes is actually deleted in Figure 13. The result of Figures 12 and 13 indicates that the improved CKAN correctly provides the individual resource deletion process of Scenario 1.

The evaluation result for Scenario 2 verified that when a data provider deletes an entire dataset, and a platform administrator performs the purge function, all the corresponding resource files are correctly deleted from Filestore and Datastore. To this end, we evaluated Scenario 2 by deleting the dataset and the resources of Scenario 1. The execution log in Figure 14 shows that the dotted-line resource in Figure 12 was deleted from Filestore. The five resources located above the solid-line resource in Figure 13 were also deleted from Datastore. In addition, Figure 15 shows that the related resources were correctly deleted by comparing Datastore tables before and after dataset deletion. The result of Figures 14 and 15 indicates that the improved CKAN correctly provides the dataset deletion process of Scenario 2.

5.2 | Evaluation of the improved harvest extension

In this section, we evaluate the improvement of Harvest Extension by experiments. First, we experimented on the new



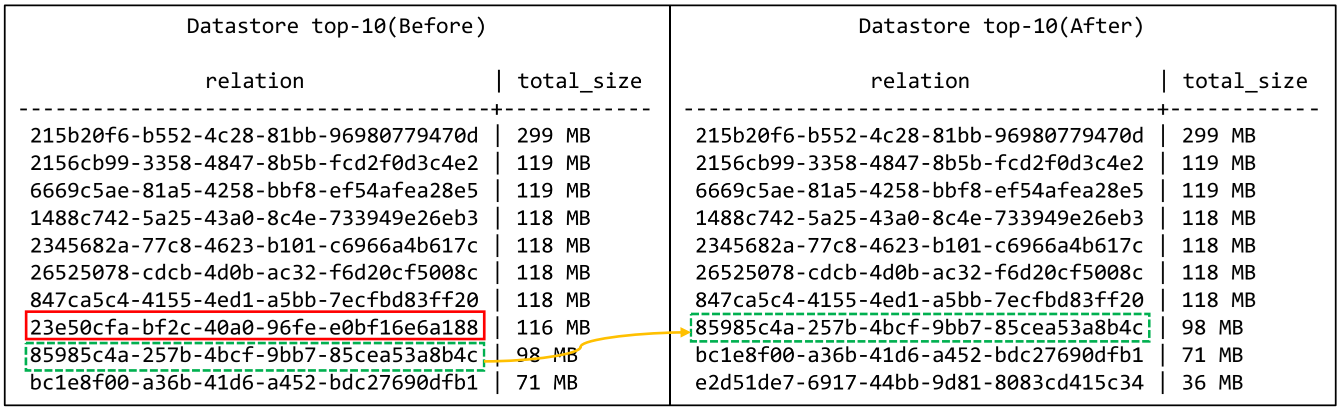
```
                                  Resource Table(Before)

              id                     |            name           | format | state  |            extras
-------------------------------------+---------------------------+--------+--------+-------------------------------
 23e50cfa-bf2c-40a0-96fe-e0bf16e6a188 | financial_advisers_1.xlsx | XLSX   | active | {"datastore_active": true}
 26525078-cdcb-4d0b-ac32-f6d20cf5008c | financial_advisers_2.xlsx | XLSX   | active | {"datastore_active": true}
 1488c742-5a25-43a0-8c4e-733949e26eb3 | financial_advisers_3.xlsx | XLSX   | active | {"datastore_active": true}
 847ca5c4-4155-4ed1-a5bb-7ecfbd83ff20 | financial_advisers_4.xlsx | XLSX   | active | {"datastore_active": true}
 6669c5ae-81a5-4258-bbf8-ef54afea28e5 | financial_advisers_5.xlsx | XLSX   | active | {"datastore_active": true}
 2345682a-77c8-4623-b101-c6966a4b617c | financial_advisers_6.xlsx | XLSX   | active | {"datastore_active": true}
 8897634c-e0e9-497d-83a9-a367bef40dcd | wexford_districts.rdf     | RDF    | active | {"datastore_active": false}

                                  Resource Table(After)

              id                     |            name           | format |  state   |            extras
-------------------------------------+---------------------------+--------+----------+-------------------------------
 23e50cfa-bf2c-40a0-96fe-e0bf16e6a188 | financial_advisers_1.xlsx | XLSX   | deleted  | {"datastore_active": false}
 26525078-cdcb-4d0b-ac32-f6d20cf5008c | financial_advisers_2.xlsx | XLSX   | active   | {"datastore_active": true}
 1488c742-5a25-43a0-8c4e-733949e26eb3 | financial_advisers_3.xlsx | XLSX   | active   | {"datastore_active": true}
 847ca5c4-4155-4ed1-a5bb-7ecfbd83ff20 | financial_advisers_4.xlsx | XLSX   | active   | {"datastore_active": true}
 6669c5ae-81a5-4258-bbf8-ef54afea28e5 | financial_advisers_5.xlsx | XLSX   | active   | {"datastore_active": true}
 2345682a-77c8-4623-b101-c6966a4b617c | financial_advisers_6.xlsx | XLSX   | active   | {"datastore_active": true}
 8897634c-e0e9-497d-83a9-a367bef40dcd | wexford_districts.rdf     | RDF    | active   | {"datastore_active": false}
```

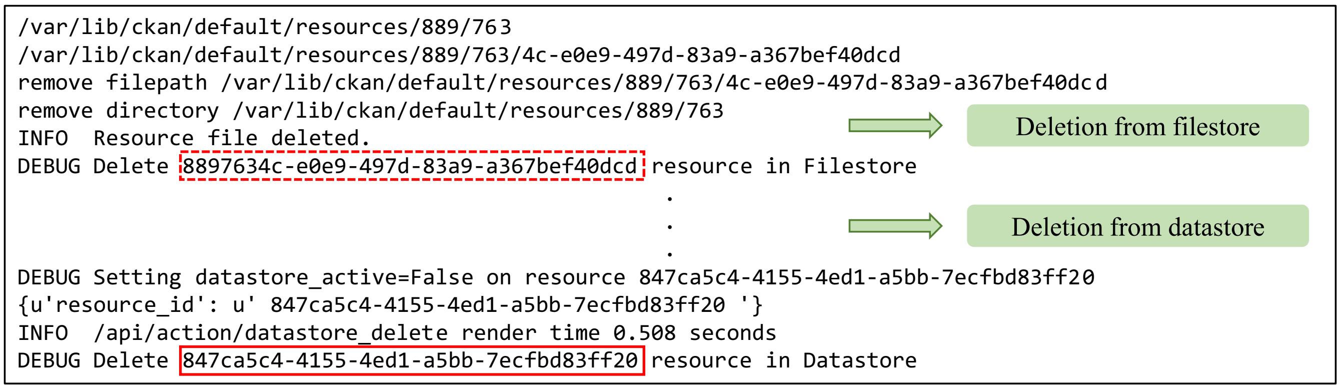
FIGURE 12 Comparison of the resource tables before and after resource deletion



```
        Datastore top-10(Before)                          Datastore top-10(After)

              relation               | total_size                relation               | total_size
-------------------------------------+-----------        -------------------------------------+-----------
 215b20f6-b552-4c28-81bb-96980779470d | 299 MB            215b20f6-b552-4c28-81bb-96980779470d | 299 MB
 2156cb99-3358-4847-8b5b-fcd2f0d3c4e2 | 119 MB            2156cb99-3358-4847-8b5b-fcd2f0d3c4e2 | 119 MB
 6669c5ae-81a5-4258-bbf8-ef54afea28e5 | 119 MB            6669c5ae-81a5-4258-bbf8-ef54afea28e5 | 119 MB
 1488c742-5a25-43a0-8c4e-733949e26eb3 | 118 MB            1488c742-5a25-43a0-8c4e-733949e26eb3 | 118 MB
 2345682a-77c8-4623-b101-c6966a4b617c | 118 MB            2345682a-77c8-4623-b101-c6966a4b617c | 118 MB
 26525078-cdcb-4d0b-ac32-f6d20cf5008c | 118 MB            26525078-cdcb-4d0b-ac32-f6d20cf5008c | 118 MB
 847ca5c4-4155-4ed1-a5bb-7ecfbd83ff20 | 118 MB            847ca5c4-4155-4ed1-a5bb-7ecfbd83ff20 | 118 MB
 23e50cfa-bf2c-40a0-96fe-e0bf16e6a188 | 116 MB            85985c4a-257b-4bcf-9bb7-85cea53a8b4c | 98 MB
 85985c4a-257b-4bcf-9bb7-85cea53a8b4c | 98 MB             bc1e8f00-a36b-41d6-a452-bdc27690dfb1 | 71 MB
 bc1e8f00-a36b-41d6-a452-bdc27690dfb1 | 71 MB             e2d51de7-6917-44bb-9d81-8083cd415c34 | 36 MB
```

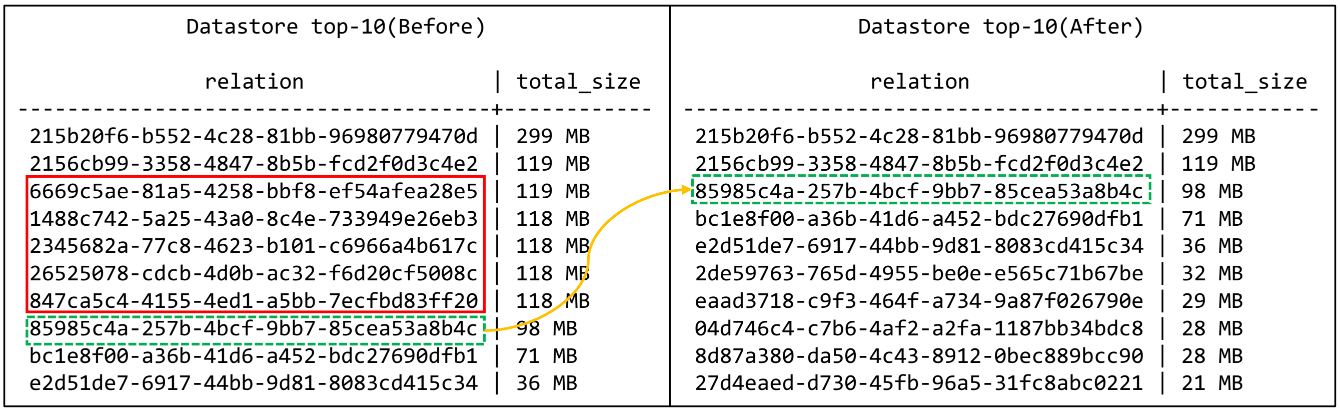
FIGURE 13 Comparison of the top 10 Datastore tables before and after resource deletion



```
/var/lib/ckan/default/resources/889/763
/var/lib/ckan/default/resources/889/763/4c-e0e9-497d-83a9-a367bef40dcd
remove filepath /var/lib/ckan/default/resources/889/763/4c-e0e9-497d-83a9-a367bef40dcd
remove directory /var/lib/ckan/default/resources/889/763
INFO  Resource file deleted.
DEBUG Delete 8897634c-e0e9-497d-83a9-a367bef40dcd resource in Filestore
                                .
                                .
                                .
DEBUG Setting datastore_active=False on resource 847ca5c4-4155-4ed1-a5bb-7ecfbd83ff20
{u'resource_id': u' 847ca5c4-4155-4ed1-a5bb-7ecfbd83ff20 '}
INFO  /api/action/datastore_delete render time 0.508 seconds
DEBUG Delete 847ca5c4-4155-4ed1-a5bb-7ecfbd83ff20 resource in Datastore
```

FIGURE 14 Resource deletion logs generated when deleting a dataset

source deletion function. The newly implemented feature deletes related data from 20 tables when the source is deleted and finally deletes source information from the harvest_source table. Figure 16 shows the output of 20 tables in the order of the largest DB size before and after applying the new deletion function. The left side of the figure depicts four harvesting-related tables before deletion, and the harvest_object table has the largest size. In the right side of the figure, the size of the



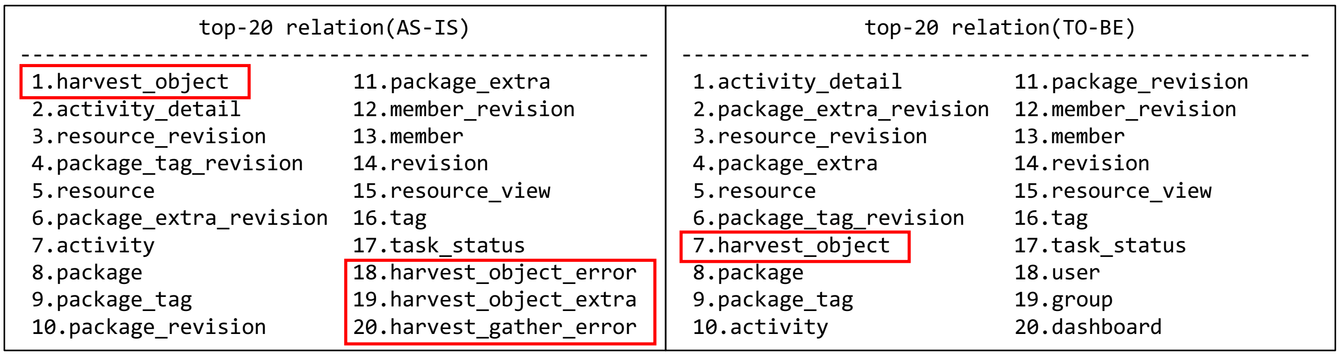
```
                Datastore top-10(Before)

                relation                  | total_size
------------------------------------------+------------
 215b20f6-b552-4c28-81bb-96980779470d     | 299 MB
 2156cb99-3358-4847-8b5b-fcd2f0d3c4e2     | 119 MB
 6669c5ae-81a5-4258-bbf8-ef54afea28e5     | 119 MB
 1488c742-5a25-43a0-8c4e-733949e26eb3     | 118 MB
 2345682a-77c8-4623-b101-c6966a4b617c     | 118 MB
 26525078-cdcb-4d0b-ac32-f6d20cf5008c     | 118 MB
 847ca5c4-4155-4ed1-a5bb-7ecfbd83ff20     | 118 MB
 85985c4a-257b-4bcf-9bb7-85cea53a8b4c     | 98 MB
 bc1e8f00-a36b-41d6-a452-bdc27690dfb1     | 71 MB
 e2d51de7-6917-44bb-9d81-8083cd415c34     | 36 MB
```

```
                Datastore top-10(After)

                relation                  | total_size
------------------------------------------+------------
 215b20f6-b552-4c28-81bb-96980779470d     | 299 MB
 2156cb99-3358-4847-8b5b-fcd2f0d3c4e2     | 119 MB
 85985c4a-257b-4bcf-9bb7-85cea53a8b4c     | 98 MB
 bc1e8f00-a36b-41d6-a452-bdc27690dfb1     | 71 MB
 e2d51de7-6917-44bb-9d81-8083cd415c34     | 36 MB
 2de59763-765d-4955-be0e-e565c71b67be     | 32 MB
 eaad3718-c9f3-464f-a734-9a87f026790e     | 29 MB
 04d746c4-c7b6-4af2-a2fa-1187bb34bdc8     | 28 MB
 8d87a380-da50-4c43-8912-0bec889bcc90     | 28 MB
 27d4eaed-d730-45fb-96a5-31fc8abc0221     | 21 MB
```

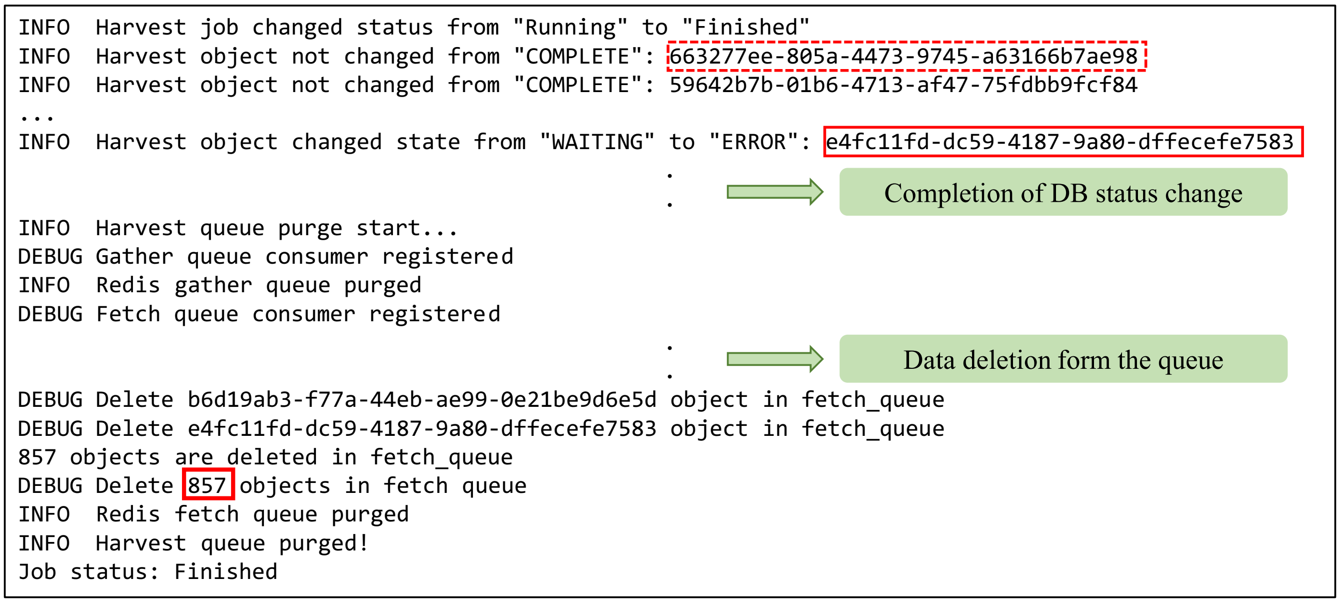
FIGURE 15 Comparison of the top 10 Datastore tables before and after dataset deletion



```
              top-20 relation(AS-IS)
----------------------------------------------------
 1.harvest_object              11.package_extra
 2.activity_detail             12.member_revision
 3.resource_revision           13.member
 4.package_tag_revision        14.revision
 5.resource                    15.resource_view
 6.package_extra_revision      16.tag
 7.activity                    17.task_status
 8.package                     18.harvest_object_error
 9.package_tag                 19.harvest_object_extra
 10.package_revision           20.harvest_gather_error
```

```
              top-20 relation(TO-BE)
----------------------------------------------------
 1.activity_detail             11.package_revision
 2.package_extra_revision      12.member_revision
 3.resource_revision           13.member
 4.package_extra               14.revision
 5.resource                    15.resource_view
 6.package_tag_revision        16.tag
 7.harvest_object              17.task_status
 8.package                     18.user
 9.package_tag                 19.group
 10.activity                   20.dashboard
```

FIGURE 16 Comparison of the DB top 20 table size before and after applying the delete function

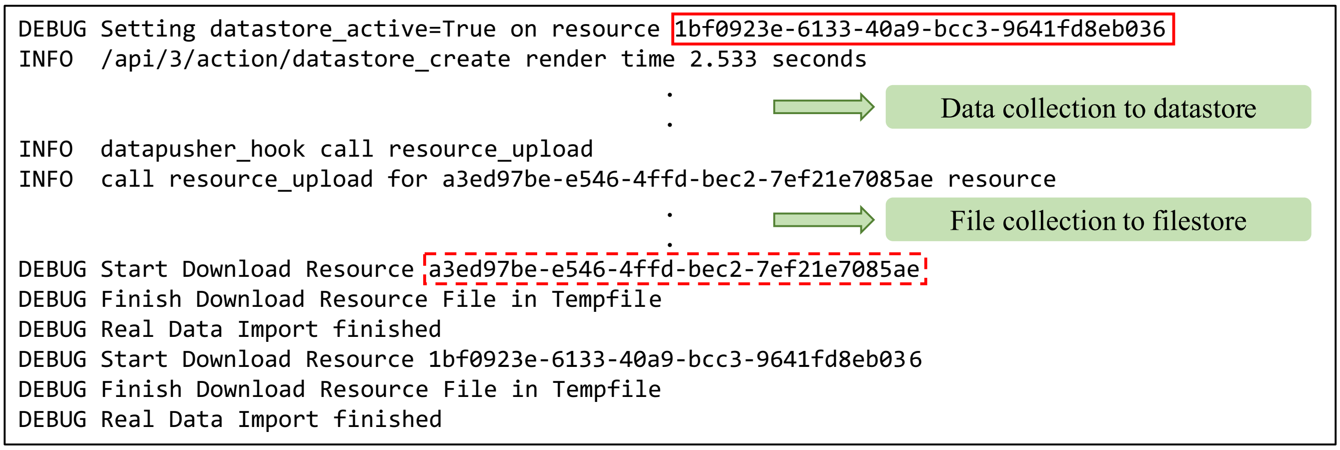


```
INFO  Harvest job changed status from "Running" to "Finished"
INFO  Harvest object not changed from "COMPLETE": 663277ee-805a-4473-9745-a63166b7ae98
INFO  Harvest object not changed from "COMPLETE": 59642b7b-01b6-4713-af47-75fdbb9fcf84
...
INFO  Harvest object changed state from "WAITING" to "ERROR": e4fc11fd-dc59-4187-9a80-dffecefe7583
                                                  .
                                                  .
INFO  Harvest queue purge start...
DEBUG Gather queue consumer registered
INFO  Redis gather queue purged
DEBUG Fetch queue consumer registered
                                                  .
                                                  .
DEBUG Delete b6d19ab3-f77a-44eb-ae99-0e21be9d6e5d object in fetch_queue
DEBUG Delete e4fc11fd-dc59-4187-9a80-dffecefe7583 object in fetch_queue
857 objects are deleted in fetch_queue
DEBUG Delete 857 objects in fetch queue
INFO  Redis fetch queue purged
INFO  Harvest queue purged!
Job status: Finished
```

FIGURE 17 Example logs when executing the new job stop function

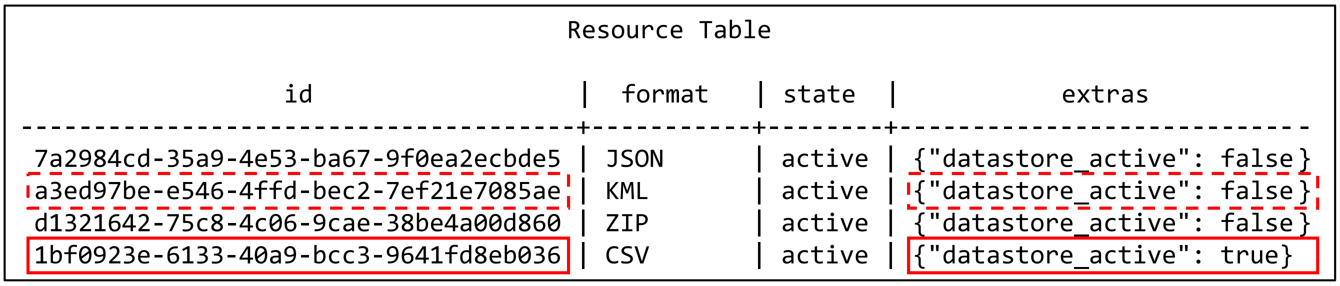
harvesting-related tables is reduced after deletion by the new feature. Only the harvest_object table remains in the top 20.

Second, we evaluated the new job stop function of the improved Harvest Extension. Figure 17 shows example logs when executing the job stop function. According to the log information, it deletes the data remaining in the queue and returns the number of deleted objects after completing the DB status change for the job and related objects. The dotted-line objects are stored as the "COMPLETE" state because their harvesting is already completed. The solid-line objects



```
DEBUG Setting datastore_active=True on resource 1bf0923e-6133-40a9-bcc3-9641fd8eb036
INFO  /api/3/action/datastore_create render time 2.533 seconds
                                                .
                                                .
INFO  datapusher_hook call resource_upload
INFO  call resource_upload for a3ed97be-e546-4ffd-bec2-7ef21e7085ae resource
                                                .
                                                .
DEBUG Start Download Resource a3ed97be-e546-4ffd-bec2-7ef21e7085ae
DEBUG Finish Download Resource File in Tempfile
DEBUG Real Data Import finished
DEBUG Start Download Resource 1bf0923e-6133-40a9-bcc3-9641fd8eb036
DEBUG Finish Download Resource File in Tempfile
DEBUG Real Data Import finished
```

FIGURE 18 Example logs showing that the improved Harvest Extension actually collects data files



```
                                  Resource Table

                  id                  | format    | state  |          extras
--------------------------------------+-----------+--------+---------------------------
 7a2984cd-35a9-4e53-ba67-9f0ea2ecbde5 | JSON      | active | {"datastore_active": false}
 a3ed97be-e546-4ffd-bec2-7ef21e7085ae | KML       | active | {"datastore_active": false}
 d1321642-75c8-4c06-9cae-38be4a00d860 | ZIP       | active | {"datastore_active": false}
 1bf0923e-6133-40a9-bcc3-9641fd8eb036 | CSV       | active | {"datastore_active": true}
```

FIGURE 19 Resource table showing the collected files by harvesting

are changed to the "ERROR" state because they are stopped before the operation is completed. The new job stop function deletes the objects in the queue after these status changes. Figure 17 shows that 857 objects are deleted from the queue.

Third, we evaluated the extended harvesting function that collects the actual data files and the metadata to determine whether the service interruption problem is resolved. Figure 18 shows example logs of the process of the Datapusher collecting the actual data files when the resource information exists during the harvesting process. In the logs, the solid-line resource is stored in Datastore, whereas the dotted-line resource is stored in Filestore because it does not match the specified format. The resource table of Figure 19 shows that the collected data files by harvesting in Figure 18 are successfully stored in the data storage. Figure 19 depicts that the solid-line resource is stored in a CSV format in Datastore, and the "datastore_active" value is marked as "true." Meanwhile, the dotted-line resource is stored in Filestore because its format is KML, indicating that the "datastore_active" value is marked as "false."

6 | CONCLUSIONS

This study proposed an improvement version of CKAN and Harvest Extension for the efficient collection and management of open data. The existing CKAN deleted metadata information only from meta-DB, and not the actual data files from Filestore and Datastore. This caused *data inconsistency* between meta-DB and data storage and *storage space waste* of storing unnecessary data files. To solve these data deletion problems of CKAN, we analyzed the operations of deleting resources, presented two deletion scenarios, and proposed a new mechanism of deleting files stored in the data storage. More specifically, we designed a new delete function of removing actual data files and implemented it in Filestore and Datastore. Using the improved deletion function, we solved the data inconsistency and storage space waste problems by deleting the actual data files from Filestore and Datastore.

From the existing Harvest Extension, we derived three problems of *source deletion*, *job stop*, and *service interruption* and improved Harvest Extension to solve those problems. First, we solved the source deletion problem by extending the deletion function to remove all relevant data from the 20 tables related to the source in addition to deleting only the source information. Second, we solved the job stop problem by improving the job stop function to not only delete the already processed data, but also clear the unprocessed data in the queue. Third, we solved the service interruption problem by extending the harvesting function to collect the actual data files in addition to the metadata such that the data service could be continued, even if the portal service is closed.

In this study, we experimentally evaluated the improved CKAN and Harvest Extension. The experimental results confirmed that the improved version correctly resolved all the problems of the existing CKAN and Harvest Extension. The results of these improvements mean that our solutions enhance the status of CKAN as an open-source platform and increase its completeness.

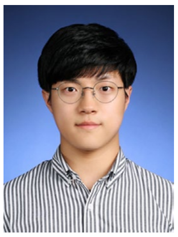
ORCID

Dasol Kim https://orcid.org/0000-0003-4389-5070
Yang-Sae Moon https://orcid.org/0000-0002-2396-0405

REFERENCES

1. CKAN documentation, available at http://docs.ckan.org/.
2. Open Government Platform (OGPL), available at https://ogpl.github.io/.
3. Socrata, available at https://dev.socrata.com/.
4. S. Corlosquet et al., *Produce and consumer linked data with drupal!*, in Proc. Int. Semantic Web Conf. (Chantilly, VA, USA), Oct. 2009, pp. 763–778.
5. Junar, available at https://www.junar.com/.
6. Open Knowledge Foundation (OKFN), why open data, available at https://okfn.org/opendata/why-open-data/.
7. A. S. Correa and F. S. Silva, *Laying the foundations for benchmarking open data automatically: A method for surveying data portals from the whole web*, in Proc. Int Conf. Dig. Gov. Res. (Dubai, United Arab Emirates), June 2019, pp. 287–296.
8. J. Winn, *Open data and the academy: An evaluation of CKAN for research data management*, in Proc. Int. Assoc. Soc. Sci. Inform. Serv. Tech. (Cologne, Germany), May 2013.
9. R. Kitchin, *The data revolution: Big data, open data, data infrastructures and their consequences*, SAGE Publications, Thousand Oaks, CA, USA, 2014.
10. F. Kirstein et al., *Linked data in the European data portal: A comprehensive platform for applying DCAT-AP*, in Proc. Int. Conf. Electron. Gov. (Tronto, Italy), July 2019, pp. 192–204.
11. B. Momjian, PostgreSQL: Introduction and Concepts, vol. 192, Addison-Wesley, Boston, MA, USA, 2001.
12. R. Copeland, Essential SQLAlchemy, O'Reilly Media, Sebastopol, CA, USA, 2008.
13. E. O'Neil, *Object/relational mapping 2008: Hibernate and the entity data model (EDM)*, in Proc. ACM SIGMOD Int. Conf. Manag. Data (Vancouver, Canada), June 2008, pp. 1351–1356.
14. D. Smiley et al., Apache solr enterprise search server, Packt, Birmingham, UK, 2015.
15. CKAN User Guide, available at https://docs.ckan.org/en/latest/user-guide.html/.
16. Jinja2 documentation, available at http://jinja.palletsprojects.com/en/2.10.x/.
17. C. Bizer, T. Heath, and T. Berners-Lee, *Linked data: The story so far,* in Semantic Services, Interoperability and Web Applications: Emerging Concepts, IGI Global, Hershey, PA, USA, 2011, pp. 205–227.
18. M. Jabalameli, M. Nematbakhsh, and A. Zaeri, *Ontology-lexiconbased question answering over linked data*, ETRI J. **42** (2020), no. 2, pp. 239–246.
19. F. Maali, J. Erickson, and P. Archer, *Data catalog vocabulary (DCAT)*, W3C Recommendation, Jan. 2014.
20. K. Banker, MongoDB in action, Manning Publications, Shelter Island, NY, USA, 2011.
21. M. Palankar et al., *Amazon S3 for science grids: A viable solution?*, in Proc. Int. workshop Data-aware Distrib. Comput. (Boston, MA, USA), June 2008, pp. 55–64.
22. C. Millette and P. Hosein, *A consumer focused open data platform*, in Proc. Int. Conf. Big Data Smart City (Muscat, Oman), Mar. 2016, pp. 1–6.
23. H. Elmekki, D. Chiadmi, and H. Lamharhar, *Open government data: Problem assessment of machine processability*, in Proc. Int. Conf. Inform. Syst. Technol. Support Learn. (Marrakech, Morocco), Oct. 2018, pp. 492–501.
24. J. J. Macedo, *OpenEasier: A CKAN extension to enhance opendata publication and management*, M.S. thesis, UFRN, Brazil, Aug. 2018.
25. A. Varon-Capera et al., *VACIT: Tool for consumption, analysis and machine learning for LOD resources of CKAN instances*, in Proc. Int. Conf. Inform. Syst. Technol. Support Learn. (Marrakech, Morocco), Nov. 2018, pp. 552–564.
26. R. Scholz et al., *A CKAN plugin for data harvesting to the hadoop distributed file system*, in Proc. Int. Conf. Cloud Comput. Serv. Sci. (Porto, Portugal), Apr. 2017, pp. 19–28.
27. D. Tunkelang, Faceted Search, Synthesis Lectures on Information Concepts, Retrieval, and Services, vol. 1, Morgan & Claypool Publishers, San Rafael, CA, USA, 2009.
28. J. Han et al., *Survey on NoSQL database*, in Proc. Int. Conf. Pervasive Comput. Appl. (Port Elizabeth, South Africa), Oct. 2011, pp. 363–366.
29. V. Ionescu, *The analysis of the performance of RabbitMQ and ActiveMQ*, in Proc. RoEduNet Int. Conf. Netw. Educ. Res. (Craiova, Romania), Sept. 2015, pp. 132–137.
30. P. Heim et al., *RelFinder: Revealing relationships in RDF knowledge bases*, in Proc. Int. Conf. Semantic Digit. Media Technol. (Graz, Austria), Dec. 2009, pp. 182–187.
31. CKAN Harvest Extension v1.1.0, available at https://github.com/ckan/ckanextharvest/releases/tag/v1.1.0/.

AUTHOR BIOGRAPHIES



Dasol Kim received his BS and MS degrees in computer science from Kangwon National University, Chuncheon, Rep. of Korea, in 2017 and 2019. He is currently a PhD student in interdisciplinary graduate program in medical bigdata convergence at Kangwon National University. His research interests include data mining, big data management, data platform, and distributed systems.

Myeong-Seon Gil received BS and MS degrees in computer science from Kangwon National University, Chuncheon, Rep. of Korea, in 2007 and 2009. From 2009 to 2011, she was a system developer at Kangwon National University, where she participated in developing mobile and web portal services. She is currently a PhD student in computer science at Kangwon National University. Her research interests include data mining, big data, and data stream analysis.

Minh Chau Nguyen received BS degree in computer science from the University of Sciences, Ho Chi Minh, Vietnam, in 2009. He received MS degree in computer science from the Korea Advanced Institute of Science and Technology, Daejeon, Rep. of Korea, in 2013. He is currently a researcher of the CyberBrain Research Section at Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea. His research interests include big data management, software architecture, and distributed systems.

Heesun Won received BS degree in computer science from Yonsei University, Seoul, Rep. of Korea, in 1990. She received her MS and PhD degrees in computer science from Korea Advanced Institute of Science and Technology, Daejeon, Rep. of Korea, in 1992 and 2016. From 1992 to 1999, she was a researcher in Korean Broadcasting System. She is currently a principal researcher of CyberBrain Research Section at Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea. Her research interests include intelligent data hub, dataset curation, and *aaS platform.

Yang-Sae Moon received his BS, MS, and PhD degrees in computer science from Korea Advanced Institute of Science and Technology, Daejeon, Rep. of Korea, in 1991, 1993, and 2001. From 1993 to 1997, he was a research engineer in Hyundai Syscomm, Inc. From 2002 to 2005, he was a technical director in Infravalley, Inc. He is currently a professor of Computer Science Department at Kangwon National University. In Kangwon National University, he was a dean in Computer Science Department from 2010 to 2013, a vice dean in Planning Department of Headquarter from 2013 to 2014, and a vice dean in College of Information Technology from 2014 to 2016. He was a visiting scholar at Purdue University, IN, USA, in 2008 to 2009. His research interests include data mining, knowledge discovery, storage systems, access methods, multimedia information retrieval, big data analysis, and data stream analysis. He is a member of the IEEE and a member of the ACM.

APPENDIX A

CKAN main tables

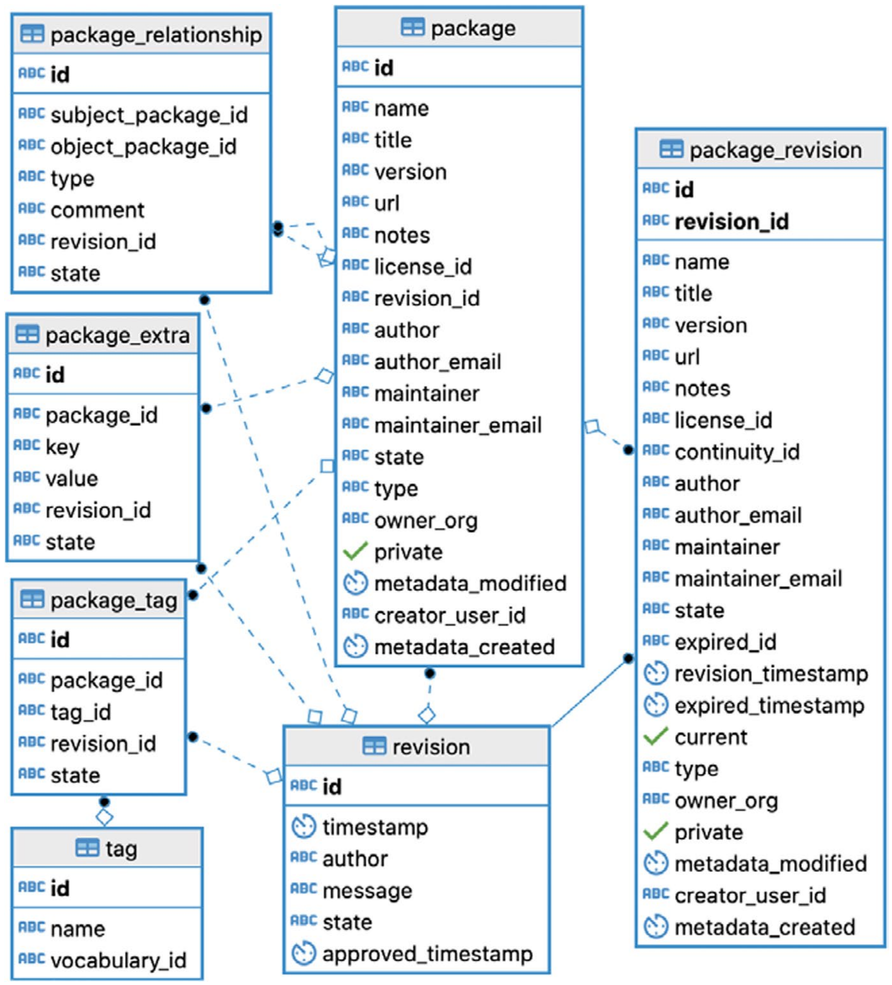
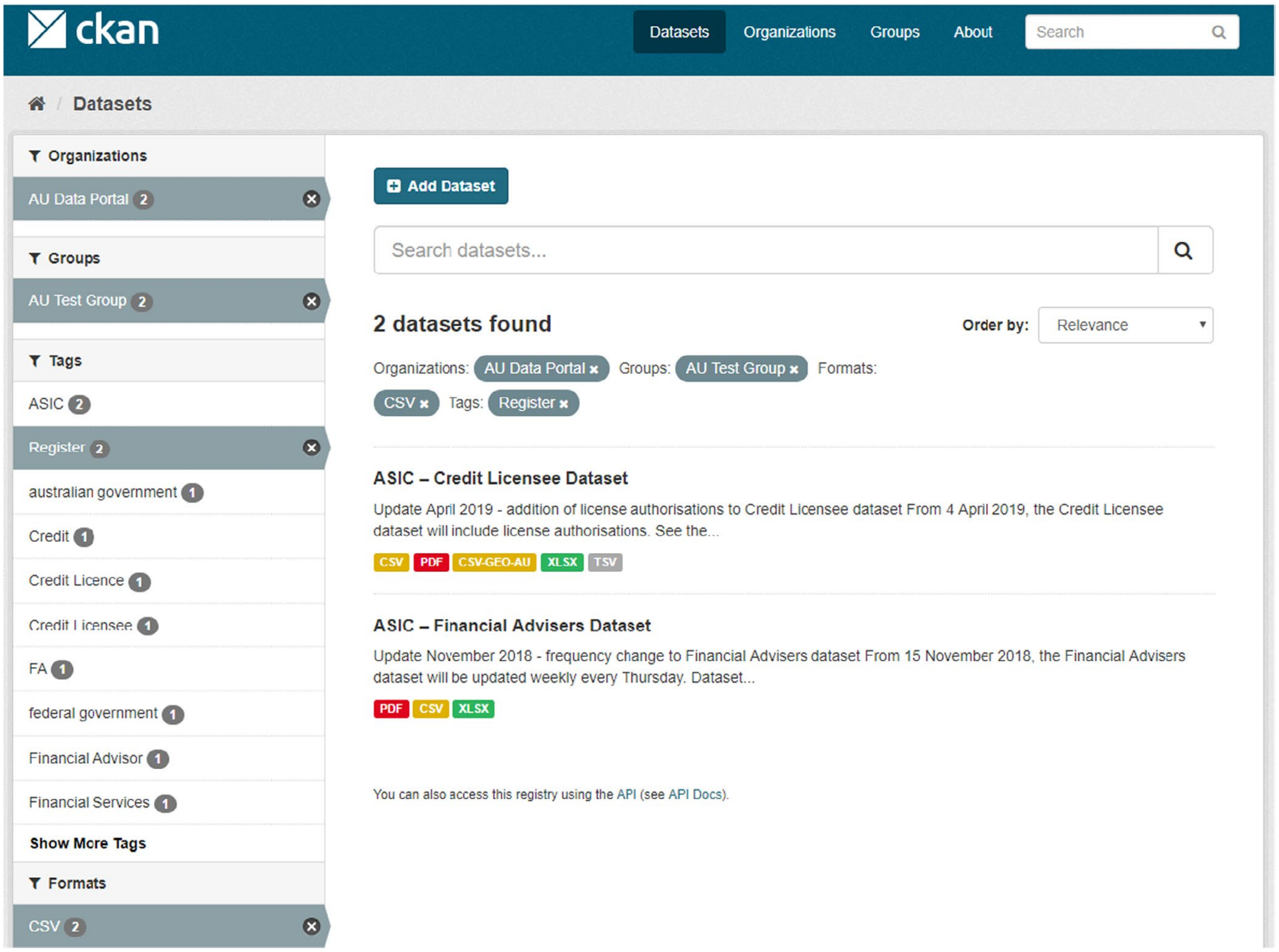


FIGURE A1 ER diagram for managing datasets in Comprehensive Knowledge Archive Network (CKAN)

Figure A1 shows an ER diagram of tables that store relevant metadata when creating a dataset. The package table stores metadata for the dataset. The package_extra table stores the custom field information given by the data provider in a key/value format. The package_tag table manages tag information for the dataset. The related tag table stores the actual tag information. The package_relationship table stores the relationship between datasets. The package_revision table stores the revision history of the data stored in the package table. Finally, the revision table is the most frequently referenced table in Comprehensive Knowledge Archive Network (CKAN), storing all the change information of the CKAN internal elements.

APPENDIX B

CKAN functions for data users



2 datasets found

Organizations: AU Data Portal Groups: AU Test Group Formats: CSV Tags: Register

ASIC – Credit Licensee Dataset

Update April 2019 - addition of license authorisations to Credit Licensee dataset From 4 April 2019, the Credit Licensee dataset will include license authorisations. See the...

CSV PDF CSV-GEO-AU XLSX TSV

ASIC – Financial Advisers Dataset

Update November 2018 - frequency change to Financial Advisers dataset From 15 November 2018, the Financial Advisers dataset will be updated weekly every Thursday. Dataset...

PDF CSV XLSX

You can also access this registry using the API (see API Docs).

FIGURE B2 Dataset exploration using search filters

Figure B2 depicts an example screenshot showing the process of searching the dataset through Faceted Search. Data users can choose one from organization, group, tag, format, and license in the taxonomy menu on the left side of the screen. The search result is presented on the right side of the screen. As described, data users can perform data preview and visualization to obtain insight on the search results.

APPENDIX C

Harvest Extension-related tables

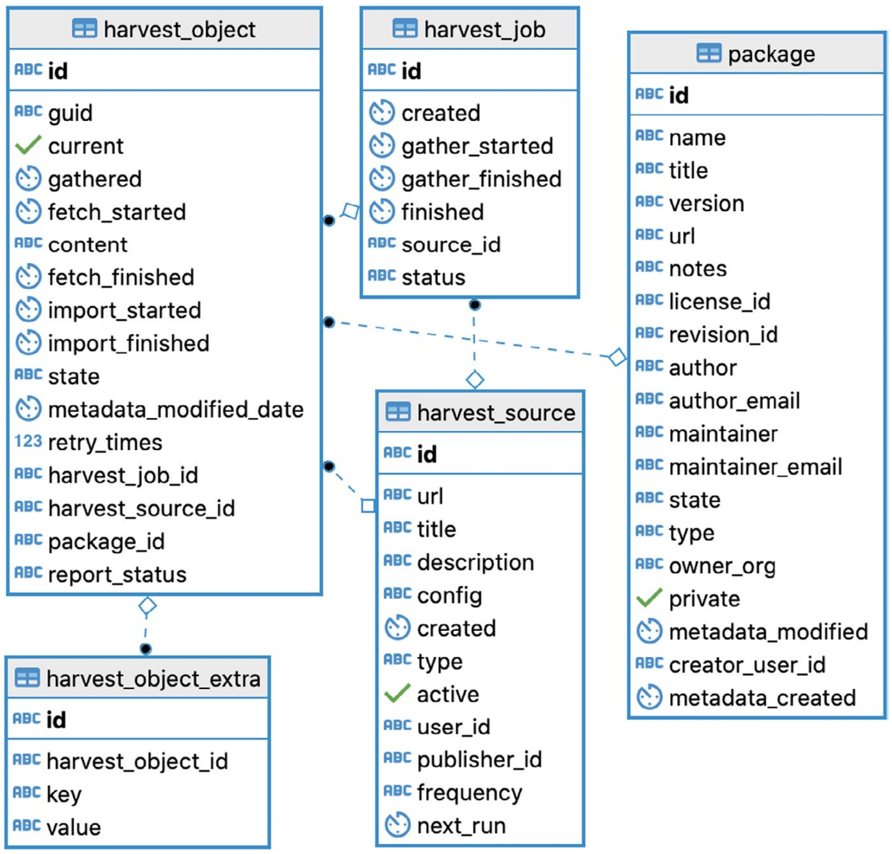


FIGURE C3 ER diagram of the tables related to Harvest Extension

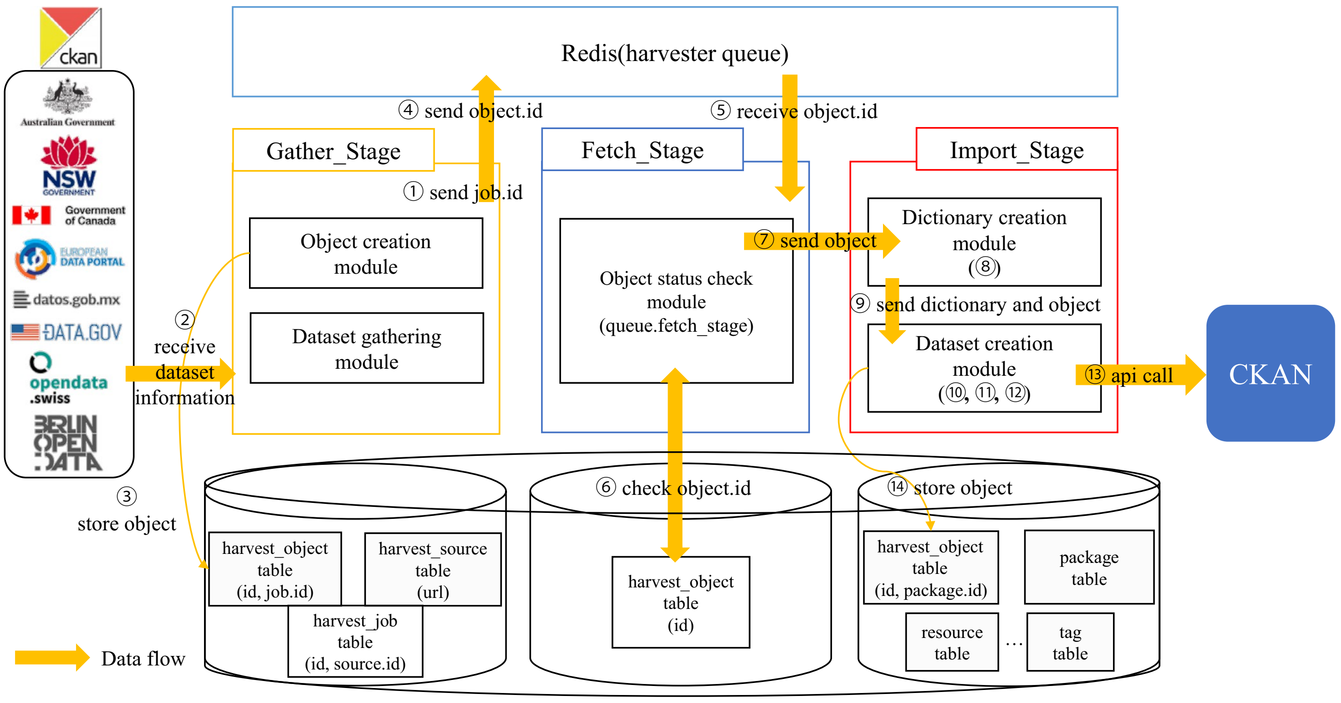
Figure C3 shows an ER diagram of the tables to be generated when Harvest Extension is added. To perform harvesting in Comprehensive Knowledge Archive Network (CKAN), we must first register the source information of the external platform having the dataset to be collected. Here, the source means a remote platform with datasets, that is, other data platforms supporting CKAN or DCAT standards. Once the source registration is completed, we run the harvest job that performs harvesting. As shown in the figure, CKAN stores the source information in the harvest_source table, job information in the harvest_job, and object information generated by the job in the harvest_object table. The object refers to the package table because it represents information generated by the dataset. Similar to the package_extra table of Figure A1, the harvest_object_extra table stores supplementary information on the object in a key/value format.

APPENDIX D

CKAN harvesting internal operation analysis

TABLE D1 Operation processes of each harvesting step

Step	Operation processes
Gather	① When a job is created, transmit the job_id to the queue
	② Check the source URL that executes the job, and read the dataset information
	③ Receive the dataset information, create an object, and store it in DB
	④ Extract the created object_id and transmit it to the queue
Fetch	⑤ Receive object_id passed through the Gather step in the queue
	⑥ Check DB to see whether the object was created normally and store the timestamp
	⑦ Transmit the entire object information to the Import step
Import	⑧ Receive an object and create a dictionary suitable for the internal dataset type
	⑨ Transmit the created dictionary and object information together
	⑩ Confirm whether the dataset is in the platform through the object and dictionary
	⑪ Compare the object's Guid with the last modification time
	⑫ Determine the object operation (skip/create/update) through a comparison
	⑬ Request the determined action to Comprehensive Knowledge Archive Network (CKAN) through API calls
	⑭ Store the final object after completing the operation inside CKAN



ckan
Redis(harvester queue)
④ send object.id
⑤ receive object.id
Gather_Stage
Fetch_Stage
Import_Stage
① send job.id
Object creation module
Dataset gathering module
Object status check module (queue.fetch_stage)
⑦ send object
Dictionary creation module (⑧)
⑨ send dictionary and object
Dataset creation module (⑩, ⑪, ⑫)
⑬ api call
CKAN
② receive dataset information
③ store object
⑥ check object.id
⑭ store object
harvest_object table (id, job.id)
harvest_source table (url)
harvest_job table (id, source.id)
harvest_object table (id)
harvest_object table (id, package.id)
package table
resource table
…
tag table
Data flow

FIGURE D4 Analysis of the detailed operations in Comprehensive Knowledge Archive Network (CKAN) harvesting

Table D1 shows the internal operations performed in harvesting with three steps. Figure D4 illustrates the operation processes of Table D1. When the user requests a harvesting job, the operations performed at each step proceed in the order described earlier. The biggest difference between the Gather, Fetch, and Import steps is that the Gather step performs operations on all objects at once, whereas the Fetch and Import steps process one object per unit.