An Automatic Collection System for Open Data in the Public Sector

Juhong Namgung, Myeong-Seon Gil, and Yang-Sae Moon
Department of Computer Science
Kangwon National University
South Korea
{namgung_juhong, gils, ysmoon}@kangwon.ac.kr

***Abstract*— In this paper, we address a problem of collecting open data from public sectors and present an automatic collection system gathering those public open data. We first analyze generic file formats and APIs that Open Data Portals used to provide the public data. Through this analysis, we then define user parameters of file- and API-based data for automatic data collection. Based on these defined parameters, we next design an automatic system that periodically collects file- or API-based data from Open Data Portals and implement the system in the real public sector environment. Finally, we verify the effectiveness of the proposed system by showing that we can successfully collect a large amount of data from three different Open Data Portals working on public sectors. We believe that the proposed automatic system makes naïve users collect and use open data of public sectors more easily and more practically.**

Keywords—data collection, automatic collection system, energy data, open data

I. INTRODUCTION

Recently, big data is being generated in various fields by the rapid spread of smart devices, IoT, and SNS applications. By these advances, there are also many research efforts and use cases on utilizing a variety of big data. In particular, by using the open big data, we can actively create many practical applications such as various demand forecasting, convenient public services, and efficient resource management. In addition, we can furthermore introduce new innovative business models by using the big data effectively and efficiently, and thus, how well we can use the open big data becomes a very important issue. Application examples of using the big data of public sectors include (1) Green Button in the United States, which allows consumers to see their energy usage and reduce energy consumption and (2) the real-time prediction of energy demand in connection with weather forecasting data [1].

As many applications and services try to actively use a variety of big data, attention and demands on data collection and deployment are also fast growing. To meet these demands, currently, many government agencies including United States, European Union, and Korea provide public sector data through their Open Data Portals. In the field of energy, Open Data Portals such as OpenEI (Open Energy Information) [2] and EIA (Energy Information Administration) [3] provide energy-related big data and show interesting use cases. These portals usually provide the big data in a variety of ways such as file downloads, data transmissions through API (Application Programming Interface), and data visualization. In particular, file downloads and API calls are the most commonly used data deployment methods. In filed-based methods, we access the files through URLs and get the files of various formats such as XML, JSON, and CSV. In API-based methods, which usually use REST APIs, we perform URL calls to get the frequently updated data in real time. However, in order to continuously collect the big data from Open Data Portals, we need to periodically download files or perform API calls, which are very manual operations and repeated jobs. In addition, APIs require different parameters for each Open Data Portal, and we thus need to make appropriate parameter settings.

To solve the problem of manual and repeated operations, in this paper, we propose a data collection system that automatically gathers open data in public sectors by supporting different file formats and APIs provided in Open Data Portals. For this, we first analyze different file formats and APIs that Open Data Portals used to provide public data. Next, we present user parameters to support those different file formats and APIs of Open Data Portals. Once the proposed system receives user parameters as inputs, it collects file- or API-based data automatically. Thus, we can continuously collect the public data with only one setup operation without repetitive manual requests or executions. Using the data type conversion function, the system also enables to save all files in CSV formats instead of XML and JSON formats. Since CSV files are very simple to use, through this conversion function, we can use the collected data more easily and practically. Finally, we implement the proposed system in the real public sector environment and show the actual results of collecting data from public sectors.

II. RELATED WORK

A. *Open Data Portal in the Public Sector*

An Open Data Portal acts as a counter for the integrated management of the open big data, making it easy for anyone to collect and use the public data. In this section, we review characteristics and use cases of several government and energy Open Data Portals.

The U.S. government created Open Data Portal (http://www.data.gov) in 2009 to build federal data

This research was supported by Korea Electric Power Corporation. (Grant number:R18XA05)

repositories between government agencies. By using the public data of the U.S. Data Portal, there are various applications such as “Where are the jobs”, which can search for salaries and statistics about various jobs at the local level, and “Roadify”, which provides real-time traffic information and notification services.

The Australian government opened Open Data Portal (http://www.data.gov.au) in 2010, and it is growing rapidly through active partnership among government agencies, industries, and private sectors. In addition, the government launched a government 2.0 task force team to pursue an open government by providing public sector data and encouraging people to participate in the public sectors more actively. It shows that the government is also interested in rapid deployment and active use of public data. Representative use cases using the open data include (1) “Suburban Trends”, which allows people to relate a city name with various indexes such as population changes and economic growth rates, and (2) “Aubiz.net”, which provides search tools to access useful information related to Australian industries and business.

For the purpose of public data deployment and policy promotion, the Korea government opened Data Portal (http://www.data.go.kr) at 2014, and the portal created communication channels for people to use a large volume of public data freely. There are also various use cases by using the big data of Korea Data Portal such as “Seoul Bus”, which a representative public mobile application that provides useful bus information in the metropolitan area, “Public Culture and Art Information”, which provides performances and exhibitions information related to culture and art hosted by public organizations, and “National Statistics Index Search”, which provides national statistics information by various tables and graphs.

OpenEI [2] and EIA [3] are the representative Open Energy Data Portals. OpenEI offers a collaboration platform based on the energy-related data. We can search for or download energy-related data and comments on the data through OpenEI. OpenEI provides not only energy data but also energy applications for data analysis. Another energy data portal, EIA, is a U.S. Energy Administration Open Portal. EIA provides a variety of energy data including oil, gas, electricity, renewable reports, and maps through file or API formats. Moreover, EIA analyzes the energy data to predict short-term and long-term trends in the energy market. One representative use case of energy-related data is the U.S Green Button [4]. The Green Button is an online platform that allows consumers to easily access their power consumption data. The Green Button informs consumers of energy usage and energy supply/demand status, and analyzes consumption patterns data to present the optimal fare plan to the users. This platform has saved 60,000 GWh of power in California, resulting in a reduction of 22.9 million tons of carbon dioxide emissions.

B. *Open Data Access Technologies*

To collect the open data from Open Data Portals, we need to use the data collection schemes like file formats and API types. Among the schemes, Open Data Portal API commonly adopts the Web API type. Most Web APIs use REST or SOAP ways, and carry data in XML or JSON formats. REST (Representational State Transfer) is a distributed computing platform model and refers to an interface for transferring data on the Web over HTTP without a separate transport layer. SOAP (Simple Object Access Protocol) is a simple XML-based protocol for exchanging data among software services in a distributed environment [5]. Most Open Data Portal APIs use REST because it is faster and easier to develop than SOAP and provides a user-friendly environment [6].

III. AN AUTOMATION OPEN DATA COLLECTION SYSTEM

Currently, many Open Data Portals deploy the open data in public sectors. However, there are many different access methods in data portals, that is, different portals provide different access methods. In this section, we analyze how to access the files and APIs to access different Open Data Portals and design an automatic collection system that autonomously gathers the open data.

A. *File-based Data*

File-based data is the most common way of providing the open data in Open Data Portals. Simply speaking, it means downloading the open data directly as a file format. According to the analysis of many portals, we know that it supports various file types such as XLS and CSV. In the file-based method, we can access the required data by simply clicking the download URL without any authentication. On the other hand, the update frequency is very low, and thus, it is widely used to deploy a large amount of accumulated static data.

B. *API-based Data*

Many Open Data Portals support Open APIs to provide frequently updated data in real time. In general, Open API means an application interface that is open for anyone to use without any authentication process or cost. In this section, we analyze the types and parameters of APIs that Open Data Portals support. We first explain how to use APIs of EIA, U.S. Energy Data Portal, as follows. To use API of EIA, we need to obtain an API key first. We then make a data request by defining the API Key, the service URL of the data, and user request parameters. The API request URL of the nuclear power generation data from EIA has the following format:

```
http://api.eia.gov/series/?api_key=YOUR_API_KEY_HERE
&series_id=TOTAL.NUETPUS.M.
```

As shown in the format above, the URL contains all the information about the request data. This request URL allows us to get the data in a JSON format. A notable point is that the API Key is mandatory, and the collection is not possible if the key is invalid. If we want to collect the data continuously at a fixed time interval, we need to periodically generate a request URL and execute API calls by considering the traffic status.

Most Open Data Portals in public sectors provide API services similar to EIA. However, the detailed methods of using APIs are different by countries. For example, in U.S., Open Data Portal does not require API Keys, and the resulting data supports various formats such as XML and CSV. In Australian Data Portal, we need to include the header information in the request URL. Because the request URL header requires a user authentication key (i.e., API Key), an error occur and data collection is impossible if the header information is missing. Fig. 1 shows an example URL for API calls to get real-time bus location from Australian Open Data Portal.

```
Curl -X GET -header ‘Accept: text/plain’ -
header ‘Authorization: apikey 3vqs****’
‘https://api.transport.nsw.gov.au/v1/gtfs/rea
ltime/buses?debug=true’
```

Fig 1. An API Request URL Example in Australian Open Data Portal.

C. *An Automation System for Collecting Open Data*

In this section, we first describe the proposed system that collects open data automatically, and we then define user parameters for the system input. Fig. 2 shows the architecture of the proposed system, and the operation sequences of the system are as follows. First, the system receives an input JSON file and saves the related parameters to determine the operation (①). The system outputs an error message and terminates if the JSON file does not match with the required input format of the system. File gathering operations are classified into file- and API-based data collections (②). If we use to collect file-based data, the system downloads the file using a URL in user parameters (②-1). In the API-based data collection, the system builds a URL based on user parameters (②-2). Through an HTTP request on the URL, we then read the required data and save the data as the file format. To distinguish and manage the collected data, we use a naming convention that combines the file name with the saved time (timestamp).

In this section, we design the proposed automatic open data collection system and define user parameters for system input. Fig. 2 shows architecture of proposed system, in which including input and output. First, system receives input from the user in order to meet user needs for automatic data collection. Input follows the JSON format and defines the data request and output.

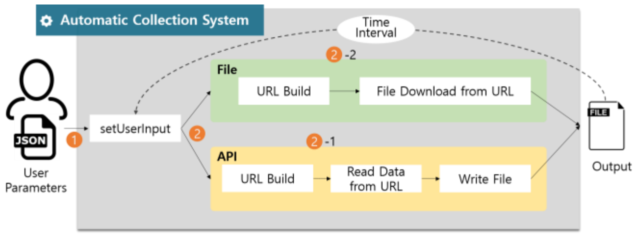


Fig. 2. An overall architecture of the automatic data collection system.

We define the user parameters of the system input as follows. In the file-based data collection, we need a file download URL with the file path and name to save the result. We can also set the time period to collect the data periodically with a given time interval. Fig. 3 shows a user parameter example using the JSON file format for the file-based data collection. `URL` in Fig. 3 means the file download URL, `directory` and `filename` specify the file path and name to store the results. Also, `timeInterval` is an optional argument for indicating periodic data collection, which presents the time period of minutes.

```
{
    "URL": " Download URL " ,
    (Option)"timeInterval": "time",
    "directory": "user_directory",
    "filename" : "user_filename"
}
```

Fig. 3. User parameter format for file-based data collection.

In API-based data collection, we also need a URL for API calls. This URL consists of a service URL, API Key, and user request parameters of Open Data Portals. Fig. 4 shows a user parameter example using the JSON file format for API-based data collection. `URL`, `serviceKey`, and `parameter` in Fig. 4 represent service URL, API Key, and user request parameters, respectively. Next, `serviceKey` and `parameter` are optional arguments. In the `serviceKey` argument, we can add URL header information through the <Key, Value> format to support different APIs of Open Data Portals. In the `parameter` argument, we can set several parameters through <Key, Value>’s because the user parameter in URL of some Open Data Portals may have multiple pairs. As in the file-based collection, we need the file path and name to store the results through `directory` and `filename` arguments. Also, `timeInterval` is an optional argument to indicate the periodic collection as in the file-based collection. In particular, in the API-based data collection, we can define the input and output file formats to support the type conversion function. The current system supports four file formats: XML, JSON, CSV, and TXT.

```
{
    "URL": " Service URL " ,
    (Option)"serviceKey": "***********",
                          | { user_key:user_value }
    (Option)"parameter": {
        "key1": "value1",
        "key2": "value2",
            …
           },
    "timeInterval": "time",
    "directory": "user_directory",
    "filename": "user_filename",
    "input_format": " xml | json | csv | txt ",
    "output_format": " xml | json | csv | txt "
}
```

Fig. 4. User parameter format for API-based data collection.

Finally, Fig. 5 is the final format of user parameters that support file- and API-based data collections. As shown in the figure, we use a `service` argument to distinguish file- and API-based data collections.

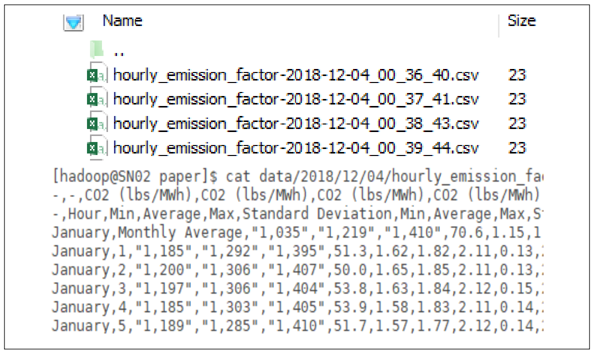
```
{
     "service": " file | api "
    "URL": " Service URL " ,
    (Option)"serviceKey": "***********",
                          | { user_key:user_value }
    (Option)"parameter": {
        "key1": "value1",
        "key2": "value2",
            …
           },
    (Option)"timeInterval": "1",
    "directory": "user_directory",
    "filename": "user_filename"
   (Option)"input_format": " xml | json | csv | txt ",
   (Option)"output_format": " xml | json | csv | txt "
}
```

Fig. 5. Final format of user parameters in file- and API-based data collections.

IV. DATA COLLECTION RESULTS

In this section, we verify the actual results of data collection from three different Open Data Portals of public sectors. The hardware specification used in the experiment is Intel i3-4150 CPU with 4GB RAM, and the software platform is CentOS 7.3 Linux operating system. We use Java to develop the collection system.

Fig. 6 shows the result of collecting the hourly energy emission amount for electricity generation in the United States data [7] from OpenEI. We use the file-based collection, and set the time period to one minute. We can confirm that the proposed system gathers the data every minute periodically.

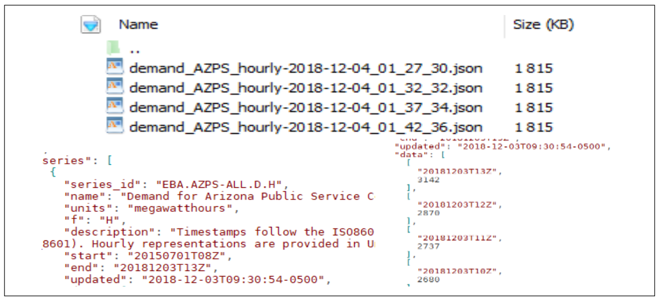


```
Name                                                Size
..
hourly_emission_factor-2018-12-04_00_36_40.csv      23
hourly_emission_factor-2018-12-04_00_37_41.csv      23
hourly_emission_factor-2018-12-04_00_38_43.csv      23
hourly_emission_factor-2018-12-04_00_39_44.csv      23
```

```
[hadoop@SN02 paper]$ cat data/2018/12/04/hourly_emission_fa
-,-,CO2 (lbs/MWh),CO2 (lbs/MWh),CO2 (lbs/MWh),CO2 (lbs/MWh)
-,Hour,Min,Average,Max,Standard Deviation,Min,Average,Max,S
January,Monthly Average,"1,035","1,219","1,410",70.6,1.15,1
January,1,"1,185","1,292","1,395",51.3,1.62,1.82,2.11,0.13,
January,2,"1,200","1,306","1,407",50.0,1.65,1.85,2.11,0.13,
January,3,"1,197","1,306","1,404",53.8,1.63,1.84,2.12,0.15,
January,4,"1,185","1,303","1,405",53.9,1.58,1.83,2.11,0.14,
January,5,"1,189","1,285","1,410",51.7,1.57,1.77,2.12,0.14,
```

Fig. 6. File-based data collection result from OpenEI.

Fig. 7 shows the result of collecting the hourly demand data for Arizona Public Service Company [8] from EIA. To show that the system supports the API-based collection as well as the file-based collection, we here use the API-based collection, and set the time period to five minutes. As in the file-based collection, we confirm that the system collects the data automatically from the portal.



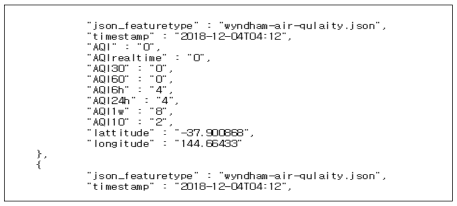
```
Name                                                Size (KB)
..
demand_AZPS_hourly-2018-12-04_01_27_30.json         1 815
demand_AZPS_hourly-2018-12-04_01_32_32.json         1 815
demand_AZPS_hourly-2018-12-04_01_37_34.json         1 815
demand_AZPS_hourly-2018-12-04_01_42_36.json         1 815
```

```
series": [
 {
   "series_id": "EBA.AZPS-ALL.D.H",
   "name": "Demand for Arizona Public Service C
   "units": "megawatthours",
   "f": "H",
   "description": "Timestamps follow the ISO860
8601). Hourly representations are provided in U
   "start": "20150701T08Z",
   "end": "20181203T13Z",
   "updated": "2018-12-03T09:30:54-0500",
```

```
"updated": "2018-12-03T09:30:54-0500",
"data": [
  [
    "20181203T13Z",
    3142
  ],
  [
    "20181203T12Z",
    2870
  ],
  [
    "20181203T11Z",
    2737
  ],
  [
    "20181203T10Z",
    2680
```

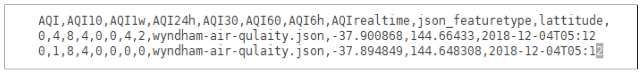
Fig. 7. API-based data collection result from EIA.

Fig. 8 shows the result of collecting Wyndham's air quality data [9] from Australian Open Data Portal. In this portal, the result data obtained by API calls has the JSON format as in Fig. 8(a). To verify the type conversion function, we set the related parameters to store the result as a CSV format. As shown in Fig. 8(b), we confirm that the system stores the data automatically in the CSV format according to the type conversion settings.



```
                "json_featuretype" : "wyndham-air-qulaity.json",
                "timestamp" : "2018-12-04T04:12",
                "AQI" : "0",
                "AQIrealtime" : "0",
                "AQI30" : "0",
                "AQI60" : "0",
                "AQI6h" : "4",
                "AQI24h" : "4",
                "AQI1w" : "8",
                "AQI10" : "2",
                "lattitude" : "-37.900868",
                "longitude" : "144.66433"
        },
        {
                "json_featuretype" : "wyndham-air-qulaity.json",
                "timestamp" : "2018-12-04T04:12",
```

(a) API call result data.



```
AQI,AQI10,AQI1w,AQI24h,AQI30,AQI60,AQI6h,AQIrealtime,json_featuretype,lattitude,
0,4,8,4,0,0,4,2,wyndham-air-qulaity.json,-37.900868,144.66433,2018-12-04T05:12
0,1,8,4,0,0,0,0,wyndham-air-qulaity.json,-37.894849,144.648308,2018-12-04T05:12
```

(b) Stored data using the type conversion function of system.

Fig. 8. API-based data collection result using the type conversion.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we addressed a problem of collecting public data from Open Data Portals. It was a very repetitive and manual operation to collect the public data periodically from Open Data Portals by using file downloads and API calls. To solve this problem, we first analyzed recent data deployment methods and defined user parameters that supported different file formats and APIs of Open Data Portals. We then proposed an automatic system that periodically gathered open data using user parameters as the system input. The proposed system collected file- and API-based public data continuously from Open Data Portals and supported the type conversion function for the resulting data. To prove the effectiveness of the proposed system, we gathered the data from different Open Data Portals of public sectors and confirmed that the system successfully collected the open data based on the given user parameters. Thus, we believe that the proposed system makes naïve users collect and use various open data of public sectors more easily and practically.

REFERENCES

[1] Y.-H. Kwak, S.-H. Cheon, C.-Y. Jang, and J.-H. Huh, "Real-time energy demand prediction method using weather forecasting data and solar model," *Korean Journal of Air-Conditioning and Refrigeration Engineering*, Vol. 25, No. 6, pp.310-316, June 2013 (in Korean).

[2] Open Energy Information, http://openei.org/.

[3] U.S. Energy Information Administration, http://www.eia.gov/.

[4] Green Button, http://www.energy.gov/data/green-button/.

[5] D.-S. Chun, S.-J. Cha, K.-O. Kim, and K.-C. Lee, "Development of efficient search engine for web services and openAPIs by keyword", In *Proc. of the Korean Information Science Society Conference*, Vol. 35, No. 1(C), pp. 159-164, June 2008 (in Korean).

[6] Y. M. Park, A. K. Moon, H. K. Yoo, Y. C. Jung, and S. K. Kim, "SOAP-based web services vs. RESTful web services," *Electronic and Telecommunications Trends*, Vol. 25, No. 2, pp. 112-120, Apr. 2010 (in Korean).

[7] Hourly Energy Emission Factors for Electricity Generation in the United States, http://openei.org/datasets/dataset/hourly-energy-emission-factors-for-electricity-generation-in-the-united-states/.

[8] Electric Demand for Arizona Public Service Company Data, https://www.eia.gov/opendata/qb.php?category=2122628&sdid=EBA.AZPS-ALL.D.H/.

[9] Wyndham Air Quality, http://data.gov.au/dataset/wyndham-air-quality/.