Alan Turing Institute – Scottish Enterprise Analysing Humanities Data using Cray Urika-GX

Rosa Filgueira, Mike Jackson

Alan Turing Institute / EPCC, The University of Edinburgh

31st July 2018

1. Introduction

In this report we describe work done to date in conjunction with Melissa Terras, College of Arts, Humanities and Social Sciences (CAHSS)[1], The University of Edinburgh. This work looked at deploying data hosted by CAHSS within the Alan Turing Institute's Cray Urika-GX system and running text analysis codes, developed by University College London (UCL), upon these. The codes were suggested by both Melissa and Raquel Alegre, of Research IT Services, at UCL. The motivation for this work was that the data and codes would serve as a real-world example to exercise the Alan Turing Institute's Cray Urika-GX system's data transfer and data analysis services.

This work was funded by Scottish Enterprise as part of the Alan Turing Institute-Scottish Enterprise Data Engineering Program.

2. About the Cray Urika-GX system

The Cray Urika GX system [2] [3] [4] is a high-performance analytics cluster with a pre-integrated stack of popular analytics packages, including Apache Spark[5], Apache Hadoop and Jupyter notebooks[6], all managed using the Apache Mesos[7] resource manager. These are complemented with myriad tools and frameworks to allow data analytics applications to be developed in Python, Scala, R and Java.

The Alan Turing Institute's deployment of the Cray Urika GX system (hereon termed Urika) includes 12 compute nodes (each with 2x18 core Broadwell CPUs), 256GB of memory and 60TB of storage (within a Lustre high-performance parallel file system) and 2 login nodes. Both compute and login nodes run the CentOS 7.4 operating system.

---

[1] https://www.ed.ac.uk/arts-humanities-soc-sci
[2] https://www.cray.com/products/analytics/urika-gx
[3] https://ati-rescomp-service-docs.readthedocs.io/en/latest/cray/introduction.html
[4] https://www.epcc.ed.ac.uk/facilities/other-facilities/cray-urika-gx
[5] https://spark.apache.org
[6] http://jupyter.org
[7] http://mesos.apache.org

3. Data

The data, hosted at the University of Edinburgh within the University's DataStore[8] provided by Information Services' Research Data Services, are as follows.

3.1. British Library Newspapers

The British Library Newspapers data [9] [10] is from Gale[11], a division of CENGAGE[12] . The data is licenced and has restrictions on how it can be shared. Within the University, the data is managed by the University Library[13]. The complete data consists of ~1TB of digitised versions of newspapers from the 18th to the early 20th century. Each newspaper has an associated folder of XML documents where each XML document corresponds to a single issue of the newspaper. Each XML document conforms to a British Library-specific XML schema.

3.2. British Library Books

The British Library Books data[14] [15] is also provided by Gale. The data is available under an open, public domain, licence. Within the University, the data is managed by CAHSS[16].

The complete data consists of ~1TB of digitised versions of ~68,000 books from the 16th to the 19th centuries[17]. The books have been scanned into a collection of XML documents. Each book has one XML document one per page plus one XML document for metadata about the book as a whole. The XML documents for each book are held within a compressed, ZIP, file. These ZIP files occupy ~224GB.

The ZIP files are collected into directories, corresponding to time periods, as shown Figure 1.

```
1510_1699/
1700_1799/
1800_1809/
1810_1819/
1820_1829/
1830_1839/
1840_1849/
1850_1859/
1860_1869/
1870_1879/
```

[8] https://www.ed.ac.uk/information-services/research-support/research-data-service/working-with-data/data-storage
[9] https://www.ed.ac.uk/information-services/library-museum-gallery/finding-resources/library-databases/databases-subject-a-z/database-newspapers
[10] The data is in 5 parts e.g. Part I: 1800-1900, https://www.gale.com/uk/c/british-library-newspapers-part-i. For links to all 5 parts, see https://www.gale.com/uk/s?query=british+library+newspapers.
[11] https://www.gale.com
[12] https://www.cengage.com/
[13] DataStore location: \\sg.datastore.ed.ac.uk\sg\lib\groups\lac-store\blpaper
[14] https://www.bl.uk/collection-guides/datasets-for-content-mining
[15] https://figshare.com/articles/BL_Labs_Flickr_Data/1269249
[16] DataStore location: \\chss.datastore.ed.ac.uk\chss\chss\groups\Digital-Cultural-Heritage
[17] The web pages and documentation for the data state the data is from the 17th to 19th centuries. However, one of the data directories spans the period 1510-1699.

```
1880_1889/
1890_1899/
```

Figure 1: British Library Books data directories in BritishLibraryBooks/ directory

Each directory has ZIP files with the XML documents for each book from that period, as shown in Figure 2. Note that the ZIP file name includes the number of pages in the book.

```
000000874_0_1-22pgs__570785_dat.zip
000001143_0_1-20pgs__560409_dat.zip
000051983_0_1-92pgs__568584_dat.zip
000059805_0_1-56pgs__579101_dat.zip
000075704_0_1-44pgs__1083069_dat.zip
000106622_0_1-84pgs__569290_dat.zip
000109780_0_1-64pgs__574740_dat.zip
000111406_0_1-66pgs__574745_dat.zip
000159302_0_1-96pgs__574909_dat.zip
000188917_0_1-76pgs__578004_dat.zip
...
004115210_0_1-76pgs__581006_dat.zip
```

Figure 2: A subset of the 693 ZIP files for books in the period from 1510 to 1699

Each ZIP file for a book contains both its XML metadata file and, in an "ALTO/" subdirectory, its XML files, one per page. See, for example, Figure 3.

```
004115210_metadata.xml
ALTO/:
004115210_000001.xml
004115210_000002.xml
004115210_000003.xml
004115210_000004.xml
004115210_000005.xml
004115210_000006.xml
004115210_000007.xml
. . .
004115210_000075.xml
004115210_000076.xml
```

Figure 3: XML metadata file and XML files, one per page, from 004115210_0_1-76pgs__581006_dat.zip

The metadata file contains metadata about the book as a whole, such as title, year, editor, issue date etc. This is represented in as a Metadata Encoding and Transmission Standard (METS)[18] XML document. See, for example, Figure 4.

[18] http://www.loc.gov/standards/mets/

```
<?xml version='1.0' encoding='UTF-8'?>
<mets xmlns="http:/www.loc.gov/METS/" xmlns:MODS="http:/ww
w.loc.gov/mods/v3"><dmdSec ID="MODSMD">
    <mdWrap LABEL="Bibliographic meta-data of the printed version" MDTYPE="MODS"
MIMETYPE="text/xml">
      <xmlData>
        <MODS:mods version="3.1">
          <MODS:titleInfo>
            <MODS:title>[A Joviall Crew: or, the Merry Beggar. Presented in a
comedie, etc.]</MODS:title>
          </MODS:titleInfo>
          <MODS:name type="personal">
            <MODS:namePart>BROME, Richard.</MODS:namePart>
            <MODS:role>
              <MODS:roleTerm authority="marcrelator"
type="text">creator</MODS:roleTerm>
            </MODS:role>
          </MODS:name>
          <MODS:typeOfResource>text</MODS:typeOfResource>
          <MODS:originInfo>
            <MODS:place>
              <MODS:placeTerm type="text">London</MODS:placeTerm>
            </MODS:place>
            <MODS:publisher>For Joseph Hindmarsh</MODS:publisher>
            <MODS:dateIssued>1684</MODS:dateIssued>
            <MODS:edition>[Another edition.]</MODS:edition>
            <MODS:issuance>monographic</MODS:issuance>
          </MODS:originInfo>
          <MODS:physicalDescription>
            <MODS:extent>59 p. ; 4°.</MODS:extent>
          </MODS:physicalDescription>
          <MODS:relatedItem>
            <MODS:titleInfo>
              <MODS:title>A Joviall Crew: or, the Merry Beggar. Presented in a
comedie, etc</MODS:title>
            </MODS:titleInfo>
            <MODS:originInfo>
              <MODS:publisher>J. Y., for E. D. &amp;amp; N. E.:London, 1652.
4°.</MODS:publisher>
            </MODS:originInfo>
            <MODS:identifier type="local">(Uk)MP1.0004404808</MODS:identifier>
          </MODS:relatedItem>
          <MODS:location>
            <MODS:physicalLocation>British Library HMNTS
644.g.23.</MODS:physicalLocation>
```

```
          </MODS:location>
          <MODS:recordInfo>
            <MODS:recordContentSource authority="marcorg">UK
</MODS:recordContentSource>
            <MODS:recordCreationDate encoding="w3cdtf">1997-06-
10</MODS:recordCreationDate>
            <MODS:recordChangeDate encoding="w3cdtf">1997-06-
10</MODS:recordChangeDate>
            <MODS:recordIdentifier>004115210</MODS:recordIdentifier>
          </MODS:recordInfo>
          <MODS:accessCondition type="Copyright Status">Outof
Copyright</MODS:accessCondition>
        </MODS:mods>
      </xmlData>
    </mdWrap>
  </dmdSec>
  </mets>
```

Figure 4: "004115210_metadata.xml" metadata file

In Figure 4, we have highlighted the year ("dateIssued") from this book. In theory, all the books stored under a directory period (e.g. "1510_1699/"), should have an issue date that belongs to that period. As we describe later (see section 7.1.1) this is not guaranteed to be the case.

The XML documents for each page are conformant with the ALTO (Analysed Layout and Text Object)[19] format, an XML Schema that details technical metadata for describing the layout and content of physical text resources, such as pages of a book or a newspaper.

4. Text analysis codes

The three text analysis codes used were as follows. The codes were initially developed by UCL with the British Library as part of Jisc Research Data Spring 2015[20] [21] and were suggested by Melissa and Raquel.

i_newspaper_rods[22] is a Python code that uses the Apache Spark cluster computing framework to run queries over newspaper data from the Times Digital Archive (TDA)[23], which, like the British Library Newspapers data is provided by Gale. It is designed to extract TDA data held within a UCL

[19] https://www.loc.gov/standards/alto/

[20] Partners in time: HPC opens new horizons for Humanities research, Research IT Services, University College London, 24 October 2017, http://www.ucl.ac.uk/research-it-services/case-studies-pub/2017-10-digital-humanities-HPC

[21] Melissa Terras, James Baker, James Hetherington, David Beavan, Martin Zaltz Austwick, Anne Welsh, Helen O'Neill Will Finley, Oliver Duke-Williams, Adam Farquhar (2017) Enabling complex analysis of large-scale digital collections: humanities research, high-performance computing, and transforming access to British Library digital collections, Digital Scholarship in the Humanities, fqx020, 02 May 2017. https://doi.org/10.1093/llc/fqx020.

[22] https://github.com/UCL/i_newspaper_rods

[23] https://www.gale.com/uk/c/the-times-digital-archive

deployment of the data management software, iRODS[24], and run queries on a user's local machine or on UCL's high performance computing services, Legion and Grace[25]. A range of queries are supported e.g. count the number of articles per year, count the frequencies of a given list of words, find expressions matching a pattern. i_newspaper_rods was last updated 1 month ago.

cluster-code (master branch)[26] (hereon called cluster-code-master) is a Python code this uses the mpi4py[27] Python wrapper for the Message Passing Interface (MPI) parallel computing protocol to query the British Library Books data. Like i_newspaper_rods it is designed for use with UCL's data management and high performance computing services. A range of queries are supported e.g. count the total number of pages across all books, count the frequencies of a given list of words, find the locations of figures etc. cluster-code-master was last updated in 2015.

cluster-code (sparkrods branch) [28] (hereon called cluster-code-sparkrods) is a branch of the above. It has been updated use Apache Spark instead of MPI. Currently only one query, to count total number of words across all books, has been updated to run under Apache Spark. cluster-code-sparkrods was last updated in 2016.

An additional code, **visualisations**[29] (hereon called visualisations) was also used. This includes Jupyter notebooks to visualise and further analyse the outputs from cluster-code-master. It also includes sample query results. Visualisations was last updated in 2015.

5. Accessing the data from within Urika

The University suggests 4 approaches to transferring[3031] data hosted within the DataStore to Linux-based systems, such as Urika:

- An NFS mount, requiring administrator access, and registration of the accessing machine with the Edinburgh Compute Data Facility (ECDF), which hosts the DataStore.
- A CIFS mount, requiring administrator access.
- An SSHFS mount.
- An SFTP file transfer, copying the data across from the DataStore.

Urika supports the following data transfer approaches[32]:

- An SFTP file transfer.
- An SCP file transfer.

We felt that SSHFS[33] mount would be preferable for the following reasons. We could mount the DataStore directories in our home directories in Urika without the need for administrators to do this for us (unlike for NFS or CIFS). Administrators would not need to register Urika with the ECDF nor set up access control to prevent unauthorised access to the data from within Urika, once mounted

[24] https://irods.org/
[25] http://www.ucl.ac.uk/research-it-services/research-computing
[26] https://github.com/UCL-dataspring/cluster-code/tree/master
[27] http://mpi4py.scipy.org/docs/
[28] https://github.com/UCL-dataspring/cluster-code/tree/sparkrods
[29] https://github.com/UCL-dataspring/visualisations
[30] https://www.wiki.ed.ac.uk/display/ResearchServices/DataStore+-+Linux+Access
[31] https://www.wiki.ed.ac.uk/display/ResearchServices/DataStore+-+SFTP+Access
[32] https://ati-rescomp-service-docs.readthedocs.io/en/latest/cray/data-transfer.html
[33] https://en.wikipedia.org/wiki/SSHFS

(unlike for NFS). We would not need to transfer all the data in one go but selectively if required, via the standard Unix file move ("mv") or copy ("cp") commands (unlike SFTP or SCP).

SSHFS was not available on Urika. We requested that it be installed and, after a discussion and an evaluation of SSHFS by the administrators, it was installed.

5.1. Copying data to Urika's Lustre file system

Before analysing the data, it must be copied from a home directory into Urika's Lustre file system ("/mnt/lustre/"). This is because, unlike Urika's login nodes, Urika's compute nodes have no network access and so cannot access the DataStore via the mount point. Equally importantly, for efficient processing, data movement and network transfers need to be minimised.

Consequently, depending on the nature and scale of data to be analysed there are two options open to users:

1. Copy the entire data set into Lustre, then run the analysis.
2. Decide how the data can be split into subsets (e.g. the British Library Books data can be split into one subset per time period), then, for each subset, copy the subset to Lustre, run the analysis on the subset, then remove the subset. When all subsets have been analysed, combine the results of each subset.

6. Using i_newspaper_rods with British Library Newspapers data

Our updated code, plus documentation on how to run it, both on a standalone machine and on Urika is available in our fork of i_newspaper_rods in an epcc-master branch[34].

6.1. Processing TDA versus British Library Books data

As mentioned in section 4, i_newspaper_rods was written to query the TDA. While the British Library Newspapers data conforms to a British Library Books-specific XML schema, the TDA XML files conform to a so-called GALEN XML Schema which specifies a document that embeds an XML sub-document with elements from the British Library Books-specific XML schema. The code within i_newspaper_rods had been written in such a way that when it queries a newspaper's XML within a document, the differences in the XML Schemas has no impact. Consequently, the i_newspaper_rods can handle both British Library Books and TDA data as-is. This may arise from the fact that the data in both cases originates from Gale.

6.2. Key changes required to i_newspaper_rods

I_newspaper_rods has two types of code:

- "Spark code": Code that can be run by Apache Spark to execute queries across XML documents.
- "fabric code": Code written using Python fabric[35] library that allows the above code plus code implementing a specific query and a file listing the locations of XML files across which the query is to be run (a so-called OIDS file) to be submitted to specific UCL systems. This

[34] https://github.com/alan-turing-institute/i_newspaper_rods/tree/epcc-master. The branch was branched from master, commit 4810474395c2f854b5a679d28e5b6ab320509e7b dated Nov 30 10:02:58 2017.
[35] http://www.fabfile.org

code queries UCL's iRODS service for the locations of the XML newspaper files and constructs the OIDS file.

We wrote additional fabric code to allow for the Apache Spark code to be run "standalone", that is, without any dependence on iRODS. This code assumes that an OIDS file is created out-of-band[36].

The Spark code assumed that the OIDS file contains relative paths to the XML documents. It prefixes these with a UCL-specific URL to turn the relative paths into UCL-specific URLs where the documents could be found. This code was changed so that the Spark code could handle an OIDS file with either URLs or absolute file paths, the latter being required so that XML documents within the Lustre file system can be accessed.

One result of this change is that the epcc-master branch will no longer work with UCL's systems. This could be fixed by updating the fabric code for UCL's systems to add the UCL-specific URL prefix when constructing an OIDS file. We did not make this change ourselves as we had no way of testing it.

Our changes were tested under Mac OS, a CentOS 7.2 virtual machine and on Urika using both OIDS files with URLs and absolute file paths.

6.3. Running queries on British Library Newspapers data

We ran the "articles_containing_words" query with a list of "interesting gender words"[37] e.g. "she", he", "him", "her", "lady", "lord" etc., which returns the frequency of each word.

We ran the modified code on a single XML document, "blpaper/xmls/0000164- The Courier and Argus/0000164_19010101.xml". An excerpt of the result is shown in Figure 5.

```
1901:
- [mary, 6]
- [his, 44]
- [gerald, 1]
- [himself, 9]
- [boy, 4]
- [brother, 4]
- [queen, 11]
- [duke, 6]
```

Figure 5: Result of a query for gender-specific words

The same result was obtained under Mac OS, a CentOS 7.2 virtual machine and on Urika.

[36] A quick-and-dirty way of doing this is to use the Unix "find" command on the directory containing the newspaper XML files to get a list of the paths to them. This is documented in the epcc-master branch's README.md file.

[37] https://github.com/alan-turing-institute/i_newspaper_rods/blob/epcc-master/query_args/interesting_gender_words.txt

7. Using cluster-code-master with British Library Books data

Our updated code, plus documentation on how to run it, both on a standalone machine and on Urika is available in our fork of cluster-code in an epcc-master branch[38].

7.1. Key changes required to cluster-code-master

cluster-code-master has two types of code:

- "MPI code": Code that uses MPI to execute queries across the contents of ZIP files.
- "fabric code": Code written using Python fabric library that allows the above code plus query code and code to determine the location of directories containing the ZIP files to be submitted to specific UCL systems.

We wrote fabric code to allow for cluster-code-master to determine the location of data directories in Urika's Lustre file system and to submit the MPI code for execution via its Apache Mesos cluster manager and using mpirun[39] to execute the MPI code across the provided resources. mrun requests 1 node and mpirun requests up to 16 CPUs on that node.

A script was written to copy the British Library Books data from a mount in a user's home directory to the Lustre file system. This file excludes all "._*.zip" files found within the directories for each time period (every ZIP file has a corresponding "._" file). It is unclear where these "_*.zip" files originate but it appears they could have been created by Mac OS (perhaps as part of the process of deploying the British Library Books data onto the DataStore).

The "mean_pages" query is misnamed as it actually calculates the total number of pages. It was renamed to "total_pages" to accurately reflect what the query does.

A "total_words" query was also written.

7.1.1. Results post-processing

Melissa suggested we look at both the "diseases" and the "normaliser" queries. "diseases" searches for occurrences of the names of 13 diseases (e.g. "cholera", "tuberculosis" etc.) and returns the total number of occurrences of each name. "normaliser" builds a derived dataset (counts of books, pages and words per year) which allows us to see how these parameters change over time. Combining the results from both queries, we can examine the extent to which occurrences of the 13 diseases terms are affected by the way that the number of books published increases over the measurement period, normalising the results and preserving statistical validity.

When a query has completed, a number of data files are output for each period, where each data file corresponds to a single parallel process. See, for example, Figure 6:

```
out_1510_1699_11.yml
out_1510_1699_14.yml
out_1510_1699_6.yml
out_1510_1699_9.yml
. . .
```

[38] https://github.com/alan-turing-institute/cluster-code/tree/epcc-master. The branch was branched from master, commit 8e0d9f24f150537c1712de4cbe768bf53b7d6986 dated Thu Jul 2 08:27:53 2015.
[39] https://wiki.mpich.org/mpich/index.php/Using_the_Hydra_Process_Manager

```
out_1890_1899_0.yml
out_1890_1899_11.yml
out_1890_1899_14.yml
out_1890_1899_4.yml
out_1890_1899_6.yml
out_1890_1899_8.yml
out_1890_1899_9.yml
```

Figure 6: Sample output files after running “diseases” query

There was no code in cluster-code-master to combine these results. A “join_diseases” script was written to combine these partial results, by concatenating the data file for each process output by the “diseases” query. A “result_diseases” script was then written to post-process these joined data files and output one file per disease (each with keys corresponding to each year in which the disease is mentioned, and a value corresponding to the number of mentions of the disease, or, the number of occurrences of that disease’s name).

Similarly, a “join_normaliser” script was written to combine the outputs of the “normaliser” query and a “result_normaliser” script was written to post-process the joined data files and output a single file, with the normaliser results for each year.

Generic “join_values” and “join_lists” scripts were also written to combine the output data from the “total_books”, and “total_pages” and “total_words” queries.

Some books that belong to a specific period (e.g. 1510 to 1699) have been reprinted later. For example, the book represented by “1510_1699/001376528_0_1-68pgs__567079_dat.zip” was reprinted in 1844 and its metadata file, “001376528_metadata.xml”, shows two issues dates (see Figure 7).

```
<MODS:dateIssued>1570 [reprinted 1844.]</MODS:dateIssued>
```

Figure 7: An issue date with a reprint date

The MPI code correctly picks up the date in brackets (in this case 1844). However, when running our “join_normaliser” script this can yield an output file with repeated keys, as shown in Figure 8: .

```
1844: [396, 169900, 65610826]
1844: [1, 68, 9394]
```

Figure 8: Duplicated keys in output files

The “result_normaliser” script handles any duplicated keys like this. See, for example, Figure 9.

```
1844: [397, 169968, 65620220]
```

Figure 9: Combining duplicated keys

Similar comments apply to “join_diseases” and “result_diseases”. It would admittedly be better if the “join” scripts were rewritten so that they did not produce output files with duplicated keys.

7.2. Running queries on British Library Books data

The “total_books”, “total_pages” and “total_words” queries were run on all the books in the “1510_1699/” directory. The results were:

- “total_books”: 693. This was validated by comparing the result to the number of ZIPs in the 1510_1699/ directory, also 693.
- “total_pages” ([books, pages]): [693, 62768].
- “total_words” ([books, words]): [693, 17479341]

The queries were also run across all books. The results were:

- “total_books”: 63701. This differs from the total number of ZIPs, which is 63700. It is unclear why this arises.
- “total_pages”: [63701, 22044324]
- “total_words”: [63701, 6866559285]

We also ran both the “diseases” and the “normaliser” queries across all the books. Comparing our results to those for 4 diseases (“cholera”, “consumption”, “measles”, “whooping”) and “normaliser” revealed differences from UCL’s results (as held in visualisations, see section 7.3. Note that visualisations does not have results for the other 9 diseases). The differences are shown in Figure 10 to Figure 14 where each figure shows both values present in one results file but not the other and differences in values with the same key (in each figure, UCL’s value then our value is shown).

```
1899: 531 (visualisations/diseases/data-ucl/cholera.yml)
1900: 2 (visualisations/diseases/data-ucl/cholera.yml)
None: 128 (visualisations/diseases/data-ucl/cholera.yml)
1881: 1216 =/= 1228
1882: 770 =/= 771
1888: 490 =/= 487
```

Figure 10: “cholera.yml”

```
1899: 748 (visualisations/diseases/data-ucl/consumption.yml)
1900: 7 (visualisations/diseases/data-ucl/consumption.yml)
1917: 5 (visualisations/diseases/data-ucl/consumption.yml)
1926: 1 (visualisations/diseases/data-ucl/consumption.yml)
None: 242 =/= 1
1847: 1299 =/= 1320
1876: 659 =/= 660
1879: 888 =/= 889
1881: 1141 =/= 1150
1882: 973 =/= 978
```

Figure 11: “consumption.yml”

```
1920: 1 (visualisations/diseases/data-ucl/measles.yml)
1899: 89 (visualisations/diseases/data-ucl/measles.yml)
1900: 2 (visualisations/diseases/data-ucl/measles.yml)
```

```
1917: 1 (visualisations/diseases/data-ucl/measles.yml)
1925: 2 (visualisations/diseases/data-ucl/measles.yml)
None: 95 (visualisations/diseases/data-ucl/measles.yml)
1879: 116 =/= 120
1898: 123 =/= 122
```

Figure 12: “measles.yml”

```
1920: 1 (visualisations/diseases/data-ucl/whooping.yml)
1899: 31 (visualisations/diseases/data-ucl/whooping.yml)
None: 47 (visualisations/diseases/data-ucl/whooping.yml)
1882: 69 =/= 72
```

Figure 13: “whooping.yml”

```
1899: [990, 310503, 91999157] (visualisations/diseases/data-ucl/normaliser.yml)
1900: [91, 13698, 2139607] (visualisations/diseases/data-ucl/normaliser.yml)
1905: [3, 356, 36763] (visualisations/diseases/data-ucl/normaliser.yml)
1906: [1, 220, 29104] (visualisations/diseases/data-ucl/normaliser.yml)
1907: [2, 128, 9345] (visualisations/diseases/data-ucl/normaliser.yml)
1908: [1, 296, 44354] (visualisations/diseases/data-ucl/normaliser.yml)
1910: [2, 440, 96451] (visualisations/diseases/data-ucl/normaliser.yml)
1911: [1, 170, 13862] (visualisations/diseases/data-ucl/normaliser.yml)
1912: [2, 124, 13553] (visualisations/diseases/data-ucl/normaliser.yml)
1913: [1, 112, 1150] (visualisations/diseases/data-ucl/normaliser.yml)
1914: [2, 160, 26777] (visualisations/diseases/data-ucl/normaliser.yml)
1915: [1, 92, 14912] (visualisations/diseases/data-ucl/normaliser.yml)
1916: [5, 444, 50850] (visualisations/diseases/data-ucl/normaliser.yml)
1917: [19, 2170, 275574] (visualisations/diseases/data-ucl/normaliser.yml)
1918: [20, 2138, 246707] (visualisations/diseases/data-ucl/normaliser.yml)
1919: [14, 1338, 163377] (visualisations/diseases/data-ucl/normaliser.yml)
1920: [18, 2422, 445459] (visualisations/diseases/data-ucl/normaliser.yml)
1921: [10, 914, 168782] (visualisations/diseases/data-ucl/normaliser.yml)
1922: [11, 1336, 165256] (visualisations/diseases/data-ucl/normaliser.yml)
1923: [3, 230, 21807] (visualisations/diseases/data-ucl/normaliser.yml)
1924: [5, 540, 75406] (visualisations/diseases/data-ucl/normaliser.yml)
1925: [8, 864, 109296] (visualisations/diseases/data-ucl/normaliser.yml)
1926: [14, 926, 104134] (visualisations/diseases/data-ucl/normaliser.yml)
1927: [7, 634, 75715] (visualisations/diseases/data-ucl/normaliser.yml)
1928: [1, 74, 2846] (visualisations/diseases/data-ucl/normaliser.yml)
1938: [1, 96, 7869] (visualisations/diseases/data-ucl/normaliser.yml)
1946: [5, 306, 39688] (visualisations/diseases/data-ucl/normaliser.yml)
1847: [468, 188806, 66595090] =/= [469, 189214, 66737593]
1859: [689, 283974, 102082700] =/= [688, 283374, 101889924]
1876: [1031, 391376, 114051280] =/= [1032, 391470, 114112523]
1878: [1005, 378031, 115847818] =/= [1006, 378237, 115885429]
```

```
1879: [1039, 365924, 110948144] =/= [1040, 366420, 111123918]
1881: [883, 338936, 118970987] =/= [887, 341590, 120946929]
1882: [770, 301002, 116444056] =/= [771, 302038, 117297592]
1885: [1252, 432249, 146452834] =/= [1254, 432249, 146452834]
1888: [1280, 414576, 115987635] =/= [1280, 413394, 115296855]
1896: [1574, 507235, 138983689] =/= [1573, 506589, 138755318]
1898: [1278, 423650, 124061119] =/= [1268, 420066, 123156558]
```

Figure 14: "normaliser.yml"

The cause of this divergence is unknown but it should be noted that UCL's results have additional key-values, including for years for which we don't seem to have books (e.g. 1900 onwards).

7.3. Visualising "diseases" and "normaliser" results using visualisations

For visualising the results, we used the visualisations code. Our updated code, plus documentation on how to run it on Urika is available in our fork of visualisations, cluster-code-visualisations, in an epcc-master branch[40].

The following changes were made:

- Existing "diseases" and "normaliser" data from UCL was moved into a new directory, each file was renamed and the "diseases" data was reedited so that each file had the results for exactly one disease[41].
- Data from our own runs was added[42].
- An existing "Diseases_1.ipynb" Jupyter notebook was copied into a new file "Diseases_1-LocalPackages.ipynb" and the following updates applied:
    - Commands were added to install any required Python libraries, if these are not already available.
    - The notebook was updated to conform to the latest Jupyter notebook format (this was automatically done when we first opened the notebook on Urika's Jupyter notebook server).
    - The notebook was modified so that it would graph all the "diseases" results and all the "diseases" results after the application of the "normaliser" data.

The Jupyter notebook was successfully used to view our data within Urika's Jupyter notebook server. The notebook uses the Bokeh[43] library to visualise interactive figures. These are not rendered within the Jupyter notebook when run in Urika. However, the HTML visualisations are created and these can be copied these to another computer and opened within a web browser.

[40] https://github.com/alan-turing-institute/cluster-code-visualisations/tree/epcc-master. The branch was branched from master, commit 3ec3c54c5fa8f5175bd3b0fb0e065df627653e70 dated Thu Jul 2 12:35:41 2015.
[41] https://github.com/alan-turing-institute/cluster-code-visualisations/tree/epcc-master/diseases/data-ucl
[42] https://github.com/alan-turing-institute/cluster-code-visualisations/tree/epcc-master/diseases/data
[43] https://bokeh.pydata.org/en/latest/

8. Using cluster-code-sparkrods with British Library Books data

Our updated code, plus documentation on how to run it, both on a standalone machine and on Urika is available in our fork of cluster-code in an epcc-sparkrods branch[44].

8.1. Key changes required to cluster-code-sparkrods

The changes made to i_newspaper_rods (as described in section 6.2) were applied to cluster-code-sparkrods too.

The "mean_pages" query is misnamed as it actually calculates the total number of words. It was renamed to "total_words" to accurately reflect what the query does.

The "total_books" query was updated to be compliant with Apache Spark.

The use of down-sampling in the Spark code[45] was commented out as it always seemed to result in zero objects and, consequently, empty query results. It is unclear why this is the case.

Our changes were tested under Mac OS, a CentOS 7.2 virtual machine and on Urika using both OIDS files with URLs and absolute file paths.

The script to copy the British Library Books data from a mount in a user's home directory to the Lustre file system, written for cluster-code-master (see section 7.1), was added here too.

8.2. Running queries on British Library Books data

The "total_words" query was run on two ZIP files:

- "dch/BritishLibraryBooks/1510_1699/000000874_0_1-22pgs__570785_dat.zip"
- "dch/BritishLibraryBooks/1510_1699/000001143_0_1-20pgs__560409_dat.zip"

The result was ([books, words]) [2, 4372]. The same result was obtained under Mac OS, a CentOS 7.2 virtual machine and on Urika. The result was validated by uncompressing the ZIP files and, via the Unix "grep", search command, searching for all occurrences of the text "<String", the XML element for a word in all the XML documents for each book's page.

The "total_books" and "total_words" queries were run on all the books in the "1510_1699/" directory. The results were:

- "total_books": 693. This was validated by comparing the result to the number of ZIPs in the 1510_1699/ directory, also 693.
- "total_words" ([books, words]): [693, 17479341]

The results were also validated by comparing them to the results for cluster-code-master (see section 7.2).

[44] https://github.com/alan-turing-institute/cluster-code/tree/epcc-sparkrods. The branch was branched from sparkrods, commit 08d8bfd0a6cf37f7e4408a9475b38d6747c0cfeb dated Nov 10 20:48:57 2016.
[45] https://github.com/alan-turing-institute/cluster-code/blob/epcc-sparkrods/bluclobber/sparkrods.py

9. Options for future work

Options for future work are as follows. No further work is proposed for cluster-code-master since that code is deprecated. However, it is important to discuss the divergence between our results and UCL's results with Melissa.

9.1. i_newspaper_rods and British Library Newspapers data

Our changes to i_newspaper_rods were made to a version last updated on November 30 2017. UCL have since made a significant number of changes to the code, including reducing its complexity, removing iRODS use, and updating it to be Python 3-compliant. We could merge these most recent changes with our branch, and then contribute our changes back (especially the support for both file paths and URLs in OIDS files and removing any UCL systems-specific code from the Spark code).

It would be useful to implement a way of constructing OIDS files that is more usable than using the Unix "find" command e.g. by providing a Python script to do this, as is now done in the most recent version of i_newspaper_rods, but in a script that contains no references to UCL-specific file systems.

We could run queries across the entire British Library Newspapers data to assess the scalability and performance of the code.

9.2. cluster-code-sparkrods and British Library Books data

The future work proposed for i_newspaper_rods is also applicable here too.

An additional task would be to migrate all the remaining queries in cluster-code-sparkrods to be Apache Spark-compliant.

9.3. General

Additional sample queries, and expected results, from Melissa for both the British Library Books data and British Library Newspaper data would be useful for performing further validation of the code when running on Urika.

Following from the above, new queries from Melissa, for which she has no results at present, would be productive to explore, contributing new insights back to Melissa.

Melissa commented that 5-6 queries that will satisfy 70% of humanities requirements (e.g. find every instance of a word and return their pages; find occurrences of one word but not the other; find occurrences of two words together, in close proximity). It would be useful to enumerate these and implement them within Apache Spark.

The Apache Spark code in i_newspaper_rods and cluster-code-sparkrods were written for an older version of Apache Spark (1.5.1) and use the pyspark[46] and streaming [47] libraries. The current version of Apache Spark (2.3.1) supports Scala, Java, and Python programming languages. Spark's support for Python is more limited than for these other languages in that a reduced set of functionalities is accessible via Python. For the purposes of text mining, all the required functionality is accessible, but one could consider migrating queries and the supporting framework from Python to Scala in future e.g. to gain performance benefits.

[46] http://spark.apache.org/docs/2.2.0/api/python/pyspark.html
[47] https://spark.apache.org/docs/2.2.0/streaming-programming-guide.html

As mentioned in section 9.1, UCL are continuing to refactor i_newspaper_rods. This is being done with the intention of combining this code with cluster-code-sparkrods, so that queries across the data can be done using a common Apache Spark with a common underlying data model. We could contribute to this work.