Brown Dog: Leveraging Everything Towards Autocuration

Smruti Padhy, Greg Jansen, Jay Alameda, Edgar Black, Liana Diesendruck, Mike Dietze, Praveen Kumar,
Rob Kooper, Jong Lee, Rui Liu, Richard Marciano, Luigi Marini, Dave Mattson, Barbara Minsker,
Chris Navarro, Marcus Slavenas, William Sullivan, Jason Votava, Inna Zharnitsky, Kenton McHenry
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Email: {spadhy, mchenry}@illinois.edu

***Abstract*—We present Brown Dog, two highly extensible services that aim to leverage any existing pieces of code, libraries, services, or standalone software (past or present) towards providing users with a simple to use and programmable means of automated aid in the curation and indexing of distributed collections of uncurated and/or unstructured data. Data collections such as these encompassing large varieties of data, in addition to large amounts of data, pose a significant challenge within modern day "Big Data" efforts. The two services, the Data Access Proxy (DAP) and the Data Tilling Service (DTS), focusing on format conversions and content based analysis/extraction respectively, wrap relevant conversion and extraction operations within arbitrary software, manages their deployment in an elastic manner, and manages job execution from behind a deliberately compact REST API. We describe both the motivation and need/scientific drivers for such services, the constituent components that allow for arbitrary software/code to be used and managed, and lastly an evaluation of the systems capabilities and scalability.**

***Index Terms*—digital preservation, unstructured data, web services**

I. Introduction

Over the past decades we have seen exponential growth in the amount of digital data with the growth only increasing as it continues to become cheaper and easier to create data digitally. This continuing shift away from physical/analogue representations of information to digital forms has created a number of social, policy, and practical problems that must be addressed in order to ensure the availability of these digital assets [1]. One aspect of these problems are that of the storage, movement, and computation on large datasets, what most think of when one hears the term Big Data, i.e. problems involving large quantities of data. Another aspect involves that of indexing and finding data as well as accessing the contents of data long term, a problem involving large amounts of data but further hindered by problems involving large varieties of data. This latter problem is a significant issue for several reasons including the rapid evolution of technology, relatively short lifespans of software, commercial interests, and the ease and reward towards creating data versus curating data. As digital software and digital data have become key elements in just about every domain of science the preservability of data has become a major concern within the scientific community with regards to ensuring the reproducibility of results. This has become a particular concern for what is often referred to as the "Long-Tail" of science, spanning the vast majority of grants involving one or more graduate students and little funds for a significant data management effort (most especially post-award). Research and development addressing this second aspect has focused on preserving the execution provenance trail [2], building repositories for scientific code/tools [3], developing user friendly content management systems [4], dealing with format conversions and information loss [5, 6], building test suites [7], as well as efforts within the artificial intelligence and machine learning communities such as computer vision [8, 9] and natural language processing [10].

We focus on two of the lower level problems involved with this latter category, a lack of appropriate metadata describing the contents of files, needed to find information of interest within large collections of data, and the lack of format specifications describing how the data is laid out within a file so that one can get at its contents (e.g. 3D/depth data, pixels, text, waveforms, etc.) independent of how it is represented on the storage medium/file system. With regards to each of these there are a number of efforts, tools, and frameworks that have been built to help users curate their own data [11, 12] and access file contents or convert to a format that can then be accessed[1,2,3]. More accurately subsets of functionality towards this exists across a wide variety of software. For example with regards to file formats, conversion capabilities exists across a heterogeneous set of libraries and software (from command line tools such as the popular ImageMagick to the GUI driven software that we use every day). With regards to metadata a similar argument can be made if we for the moment relax the typical use of the term to be solely that of data describing data, data useful for searching/indexing collections of data and its contents. In this context a wide variety of tools exist that take data and analyze it for some higher level piece of derived information that is then produced (e.g. machine learning classifiers, models of all kinds, statistical software, and actual metadata extractors).

Needs for such tools permeate the day to day workflows

[1] http://www.opendocumentformat.org/

[2] https://cloudconvert.com/

[3] https://www.ps2pdf.com/

of just about everyone. Use cases span ecology, biology, civil environmental engineering, hydrology, oceanography, material science, library & information science, social science, and so forth to including the public at large. For example many communities utilize a wide variety of models to predict/simulate events, e.g. plant growth at various areas over different time frames. These models typically support data in unique formats. Efforts such as PEcAn [13], one of our use cases, aims to make it easier to connect data sources to models by providing these conversion capabilities for an extensible number of data sources (e.g. Ameriflux[4], NARR). Similarly with navigating large collections of unstructured data. For example in biology automatically classifying microscopy images of fossilized pollen (in addition to converting out of proprietary microscopy formats), in entymology the tracking of bees within a colony, identifying plant phenology, deriving data from digitized handwritten documents, in medicine classifying/tracking cells in microscopy images, extracting data from publications (i.e. tables, figures) as no other source of the data may be available, extracting data from spreadsheets of all kinds (with different internal layouts and naming conventions), labelling land coverage in satellite and Lidar data, in material science identifying failed fabrication experiments in SEM data, finding and tracking people in social science experiments, and so on. This applies to the public at large as well with the format conversion needs we deal with regularly (e.g. between document formats, image formats, video formats, etc.) and indexing needs such as finding desired files in our photo/video collections, or smarter ways of searching a folder of documents (e.g. via NLP techniques, etc). Tools to help with portions of these are everywhere. We aim to be able to leverage all of them whether they exists as libraries, command line tools, GUI applications, or web services, and make these capabilities as trivial to users/applications as possible while simultaneously combining their abilities, and perhaps preserving these tools at the same time.

In the sections below we outline the Data Access Proxy (DAP) for file format conversions and the Data Tilling Service (DTS) for metadata extraction which make up the Brown Dog[5] effort. Both exist as services/frameworks that aim to make it as easy as possible to incorporate arbitrary conversion and extraction capabilities from 3rd party software and services, connect them to obtain the union of their capabilities, scale them dynamically to meet the demands of the service, and manage them towards a robust service. Both the DAP and DTS are modeled so as to fit a role within the internet analogous to a DNS service in terms of setup and usage by other applications. In the sections below we describe the architecture of the two services, how they interact with arbitrary software, their scalability and extensibility, an evaluation of the two services, and a number of prototype client applications.

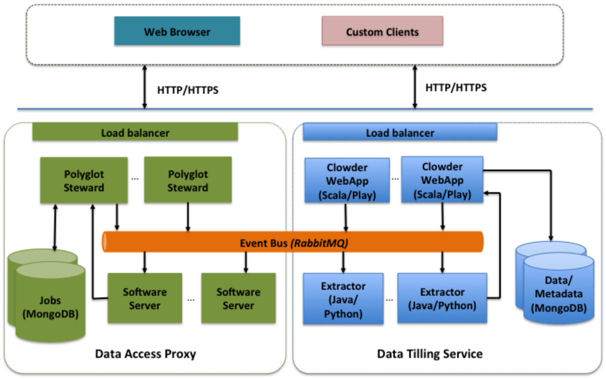


Figure 1: Architecture of the Brown Dog services. Both the DAP and DTS exist as web services behind a load balancer coordinating between a potentially distributed number of Clowder and Polyglot instances respectively. Each Clowder or Polyglot instance in turn manages a number of distributed extractors and/or Software Servers respectively which handle elements of a job.

II. Architecture

The DAP built off of the Polyglot framework [5, 14] and the DTS built on top of the Clowder framework [4, 12] are architected to manage a number of heterogeneous tools distributed across the web, specifically conversion and extraction tools respectively, and provide access to the union of their capabilities via a fairly compact REST interface (Figure 1). These tools, essentially black boxes of code/functionality, are tracked and managed by the head node services and elastically grown/shrunk to accommodate user demands. Both emphasize extensibility in the sense of allowing new converters/extractors to be added and deployed across a DAP/DTS instance as trivially as possible. A typical workflow might involve calling the DTS to index and find relevant data according to some criteria within a collection of data and using the DAP to convert the files in that collection to a format that can be processed.

A. *Data Tilling Service*

Clowder is an open source web based content management system which allows users to upload files, create datasets and collections, socially curate data by assigning tags, metadata, and leaving comments, then publishing their data to a long term archive for preservation once work with the data has been completed [12]. It also provides auto-curation through a suite of extensible extractors that are automatically triggered and executed based on appropriate file types. These extractors can do anything from pulling metadata within the file, analyzing the file's contents and tagging it according to some specific classification or criteria, towards the generation new data such as metadata, previews, sections (i.e. areas of interest), etc. Clowder's extractors exist within a cloud environment distributed across any number of physical or virtual machines and listen to a distributed messaging bus for new files to the system. The Brown Dog Data Tilling Service (DTS)[6] builds

[4] http://ameriflux.lbl.gov/

[5] Playing on the notion of a mutt.

[6] We use data tilling, like in farming, to emphasize a notion of churning data in various ways towards enhancing its usability via the uncovering of various higher level data products useful for indexing or otherwise using the data.

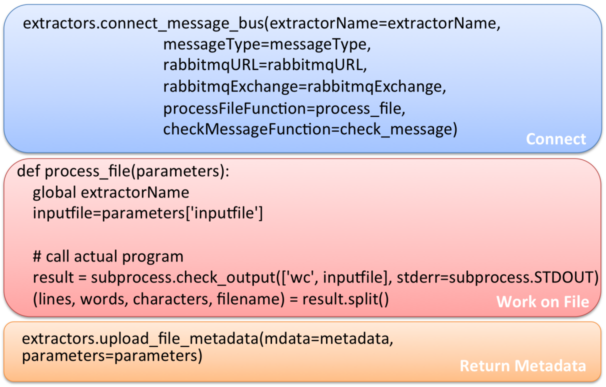
GET	/api/extractions/inputs	Lists the input file format supported by currently running extractors
POST	/api/extractions/url	Uploads a file for extraction using the file's URL
POST	/api/extractions/file	Uploads a file for extraction of metadata and returns file id
GET	/api/extractions/{id}/status	Checks for the status of all extractors processing the file with id
GET	/api/files/{id}/metadata	Gets tags, technical metadata, and content based signatures extracted for the specified file
GET	/api/extractions/extractors	Lists the currently running extractors
GET	/api/extractions/extractors/details	Lists the currently details running extractors
GET	/api/extractions/servers	Lists servers IPs running the extractors

Table 1: The DTS REST API for metadata, tags, and signature extraction.

on top of Clowder, emphasizing its REST interface towards allowing other applications to leverage its extraction capabilities, making it easier to create and deploy new extractors, building up an extensive catalogue of extractors, hardening the representation of derived data/metadata, enhancing the scalability and adding an elastic component to grow and shrink capabilities intelligently and dynamically based on user demand.

The DTS serves as a web based service where client applications or users can pass in one or more files or URLs and get back JSON or JSON-LD containing a number of derived products from tags, metadata, or other derived files that are typically higher level than the original data and/or holding some semantic information. Given this derived information applications can then use it to index, compare, and/or further analyze collections of data, in particular uncurated and/or unstructured data collections. As stated previously the main interface to the DTS is its REST API (Table 1).

At the heart of the DTS is a distributed messaging bus (Figure 1), specifically RabbitMQ[7]. A widely used and hardened framework, RabbitMQ can be used to reliably distribute and manage job execution in a distributed environment by placing messages in a queue for each extractor, taking them off when jobs are completed, and automatically resubmitting messages should a job fail. The DTS Clowder head node handles all messages to and from the distributed queue and further manages the moving and storing of intermediary files. Files uploaded to the DTS for processing are either stored or referenced and given a unique ID. At this time a message is put on the bus with the file ID along with a key based off of the files mime-type. Any extractor capable of handling that type of file takes the message off of the queue, uses the ID to obtain the file for further processing, processes the job, then returns any derived data back to Clowder associating it with the file's ID. Data and metadata stored within Clowder can be placed into one of a number of extensible storage options such as MongoDB, iRODs [15], or the local filesystem. By default we use MongoDB which is convenient in that all communication between the various Clowder components uses JSON and MongoDB is JSON document based. The underlying mongo database can further be sharded for added capacity, performance, and reliability. Lastly, the DTS Clowder head node is designed so as to be stateless allowing it to be replicated and placed behind a load balancer, e.g. NGINX, in order to increase performance and reliability.



```
extractors.connect_message_bus(extractorName=extractorName,
                    messageType=messageType,
                    rabbitmqURL=rabbitmqURL,
                    rabbitmqExchange=rabbitmqExchange,
                    processFileFunction=process_file,
                    checkMessageFunction=check_message)
```

```
def process_file(parameters):
   global extractorName
   inputfile=parameters['inputfile']

   # call actual program
   result = subprocess.check_output(['wc', inputfile], stderr=subprocess.STDOUT)
   (lines, words, characters, filename) = result.split()
```

```
extractors.upload_file_metadata(mdata=metadata,
parameters=parameters)
```

Figure 2: Creating an extractor for deployment within the DTS is simplified through the use of various language specific libraries. For example the pyClowder library allows one to create python extractors with essentially the three pieces of code shown here which: connects to the distributed queue, carries out or calls the analysis code, then lastly returns the derived data.

Extractors can be written in any language so long as it is capable of interacting with RabbitMQ and HTTP in order to access the Clowder REST interface (e.g. Java, C/C++, Scala, Python, Ruby, etc). In addition to carrying out some sort of analysis of the data, often times by wrapping some external piece of code/software, an extractor requires a relatively small amount of code in order to interact with the rest of the system. Specifically, it must register itself with the RabbitMQ bus and specify what keys it will respond too, listen for incoming messages and pick up those that it can process, and lastly return derived tags, metadata, etc. to Clowder. To further simplify the creation of extractors we have written a library, pyClowder[8], that reduces this setup to a handful of boilerplate lines of code (Figure 2). Additional libraries are in development for commonly used languages in research such as R[9] and Matlab[10].

Three types of metadata are differentiated by the system: technical metadata, versus metadata, and previews. Technical metadata is automatically generated, derived data, by the extractors, e.g. obtaining text contents within an image, classification of an object, a Greenness Index, coordinates of the specific sections of a file, etc. Versus metadata, obtained from an extractor leveraging the Versus framework [16], are the signatures extracted from a file's contents. These signatures, effectively a hash for the data, are typically numerical vectors, which capture some semantically meaningful aspect of the content so that two such signatures can then be compared using some distance measure (capturing either similarity or dissimilarity). These signatures along with Versus allow content based retrieval [9] tools to be included/provided by the DTS towards yet another means of comparing and sifting

[7] https://www.rabbitmq.com/

[8] https://opensource.ncsa.illinois.edu/stash/projects/CATS/repos/pyclowder/

[9] http://www.r-project.org/

[10] http://www.mathworks.com/products/matlab/

GET	/api/conversions/outputs	Lists all output formats that can be reached
GET	/api/conversions/inputs	List all input formats that can be accepted
GET	/api/conversions/inputs/{input format}	List all output formats that can reach the specified input format
GET	/api/conversions/outputs/{output format}	List all input formats that can reach the specified output format
GET	/api/conversions/convert/{output format}/{file URL}	Convert the specified file to the requested output format
POST	/api/conversions/convert/{output format}	Convert the uploaded file to the requested output format
GET	/api/conversions/software	List all available conversion software
GET	/api/conversions/servers	List all currently available Software Servers

Table 2: The DAP REST API for format conversions.

through data. The final type, previews, as well as other types of derived file products, are also generated by the extractors. These can include things such as thumbnails, image pyramids, sections/clips, maps, spreadsheets, etc. These derived products again aid in the navigation of data, this time in the manual human sense, perhaps providing a visualization of the data or collection of data. This may also be used as an intermediary in a chained extraction process where some other extractor will then be triggered to extract something more meaningful from this new data (e.g. generating a signature from a key frame extracted from a video). For each type of metadata the system also keeps track of the source extractor that created it for provenance purposes.

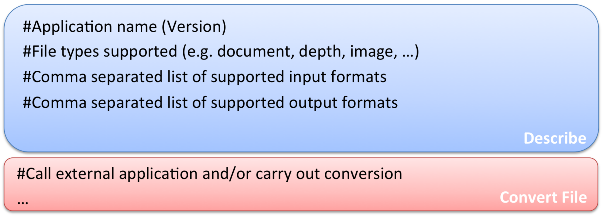
B. Data Access Proxy

The Brown Dog Data Access Proxy (DAP) handles conversions between formats, ideally to one that is more readily accessible for the user. Unlike the DTS which takes no parameters but triggers any extractor that will fire based on the file type, the DAP takes a single parameter specifying the desired output format. Like the DTS, the DAP provides a compact REST interface allowing users and applications to call its capabilities (Table 2).

Built on top of the Polyglot framework the DAP is designed to support the inclusion of any piece of code, library, software, or service into its ecosystem of conversion tools. A component tool called a Software Server [14] utilizes one or more very light weight wrapper scripts to automate specific capabilities within arbitrary code and then provide access to it via a consistent REST interface:

```
https://<host>/software/:application/:output/:file
```

Applications can then call, program against, these capabilities within the software as easily as they would a library. When GUI applications are involved scripting languages such as AutoHotKey[11] or Sikuli [17] are used to wrap needed open/save functionality. Any text based scripting language may be used for these scripts so long as it follows certain conventions in its comments indicating the name and version of the software, the types of data it handles (e.g. documents, 3D), and possesses a list of accepted input and output formats (Figure 3). The DAP also supports the use of Data Format Definition Language (DFDL) schemas [18], a relatively recently standardization of a machine readable language for format specifications, through the use of a specialized wrapper script for Daffodil, an open source implementation of DFDL. New schemas can be incorporated into the DAP by making a minor modification to this template script. DFDL schemas, generating XML representations from data contents for a given format, possibly mapping elements to standardized ontologies, provide a long term/preservable means of capturing the layout of file contents, particularly important for the many ad hoc formats scientists/graduate students utilize in their work (e.g. for tabular, spreadsheet like data, or other representations for data).



```
#Application name (Version)
#File types supported (e.g. document, depth, image, …)
#Comma separated list of supported input formats
#Comma separated list of supported output formats
```

```
#Call external application and/or carry out conversion
…
```

Figure 3: Adding converters to the DAP is done by simply annotating a wrapper script for the software in the scripts comments. The first few comments tell the DAP what the software is, what types of data it works on, and what inputs and outputs it supports. This is sufficient for the DAP to manage and delegate conversion jobs to that software. The remainder of the script either carries out or calls another application to do the conversion.

When a Software Server comes online, each potentially hosting one or more applications, it will attempt to connect to a specified RabbitMQ bus and then listen to one queue per software that it controls. The DAP Polyglot head node instance monitors the RabbitMQ bus, specifically the queues and the consumers of the queues, leveraging it as a discovery service for new Software Servers, software, and conversion capabilities. For each Software Server found, it queries it for the applications provided and the input and output formats each in turn supports. From this a graph is constructed, referred to as an input/output graph or I/O-graph, with formats as the vertices and applications as directed edges between vertices indicating conversions they are capable of carrying out. Given an input format and a desired output format the DAP will search this graph for a shortest path between the source and target format, allowing conversions to occur that would require multiple applications, possibly running on different machines. The constructed job is stored in a MongoDB instance, where one or more DAP Polyglot instances will monitor it and move it across the path, placing portions of the job on the appropriate application queues as need be. Since all information about the job is stored in a shared MongoDB instance the Polyglot head nodes are also stateless as in the case of the DTS and can be placed behind a load balancer for added performance and reliability. Software Servers will monitor the relevant software queues, pull off jobs, execute them on the local hardware, and return a link to the resulting output file back to the DAP head node. All files are preferably passed between the DAP, each Software Server, and even the external source, as URLs

[11] http://www.autohotkey.com/

in order to minimize file transfers. This saves transfers in a number of instances, e.g. returning the output of the last Software Server called to the DAP head node which then returns it to the user, or should a Software Server possess the needed applications for two parts of a conversion path.

As a cloud based service it is expected that the applications and Software Servers are elastic, i.e. new ones will come on line on occasion, and current ones will go offline on occasion. As such the I/O-graph must be updated continuously so that all currently valid conversion paths are represented. A thread within each DAP Polyglot instance will continuously poll the consumers on the RabbitMQ bus. When new Software Servers are found their vertices and edges are simply added to the graph, which represents the union of capabilities among all discovered Software Servers. When a found Software Server no longer responds, for whatever reason, the constituent edges for its applications are pruned from the graph. In order to make the traversal and the appending of new applications to the graph efficient the I/O-graph is represented as an adjacency list in memory. This however, makes the removal of edges and vertices somewhat costly, especially for large graphs, thus pruning is done sparingly and limited only to the edges which is sufficient to eliminate invalid conversion paths.

C. Elasticity

The two building blocks of Brown Dog, the DTS and DAP, which are built to utilize arbitrary code/software as extractors/converters, need to have the ability to handle heavy loads, adapt to spikes in requests, handle a mix of long running and short running jobs, and support heterogeneous architectures. Specifically, the two services need to be able to auto-scale up/down based on the demand of the system. Towards this end, we designed and implemented an elasticity module that focuses on auto-scaling the DTS's extractors and DAP's Software Servers by leveraging cloud computing Infrastructure-as-a-Service. The module could be extensible to other services as well with some modifications. Based on extractor/converters types, the module needs to support multiple operating system (OS) types, including both Linux and Windows, for its proper execution. Further we wish to support a variety of Virtual Machine (VM)/container frameworks to allow extractors and Software Servers to be deployed on a variety of different resources.

We considered a number of cloud computing software platforms, both open source such as OpenStack, Cloud Stack, and Eucalyptus; and commercial such as Amazon Web Services (AWS), Microsoft Azure, and VMWare; as well as other related technologies such as Olive, OpenVZ and Docker. AWS is relatively mature but can be costly as when CPU and memory usage go up the cost of using AWS also goes up. In the open source space, OpenStack is mature, widely adopted, and supports both Linux and Windows. For our initial setup we chose OpenStack as the top level VM technology, Unix system services as the low level technology, and Docker as an intermediary level technology candidate with regards to the level of granularity by which we control the elasticity (e.g. Docker has a smaller resource usage overhead and faster VM/container startup time compared with OpenStack).

Algorithm 1 BDMonitor()

```
Read the control parameters' values from the config file
while TRUE do
    Build OpenStack Server Map
    Build Run-time Mapping between Services and running VMs, $srvc2rVMMap$
    Obtain VMs usage information, e.g., $loadAverage$ and $numvCPU$ and build $vmInfoMap$
    ScaleUp()
    ScaleDown()
end while
```

1) Elasticity Module Design: There have been a variety of efforts with regards to elasticity in the cloud computing context [19]. Lots of the policies discussed in these works, and also the ones provided by many cloud providers, allow scaling based on criteria such as i) VM internal information e.g. CPU usage, memory, etc; ii) monitoring job queues, and iii) predictions of workload patterns.

We make the following assumptions in our design: i) an extractor or a Software Server is installed as a service in a VM/container. Extractors of the same type, i.e. requiring the same execution environments, can be deployed in the same VM so when a VM starts, all the extractors that the VM contains as services will start automatically and successfully, ii) the resource limitation of using extractors/Software Servers to process input data is largely CPU processing, iii) we wish to support deployment across any number of available cloud infrastructures, iv) the system uses RabbitMQ as the messaging technology.

From this we provide an auto scaling solution that is based on the queue lengths at the message queues for extractors and Software Servers. The DTS web application is a loosely coupled publish/subscribe system and RabbitMQ which acts as a broker provides finer grained details such as queue lengths, channel activities, connection details, consumer details etc. through it's management API. Based on these fine grained details and information obtained from the cloud infrastructure API on the VM's internal information scaling actions at two levels are performed: service-level and VM-level. At the service-level, an extra instance of the service is deployed in the VM already running the same type of services while at the VM-level a new VM or a suspended VM containing the service is started. In our algorithm, the module will monitor the message queue and based on a predefined queue length threshold will take scaling actions accordingly either using the cloud infrastructure API to start/suspend/resume/terminate a VM or to start/stop a service in a running VM based on CPU load average. As per the classifications in [19], our policy is manual reactive, and the method is a mix of horizontal (adding VMs) and vertical scaling (adding instances on a VM).

Algorithm 1 shows the pseudo code for the auto scaling that we use in our implementation which periodically checks if scaling up/down is required for the service. It reads all the parameters needed from a configuration file. Algorithm 2 and Algorithm 3 show the pseudo code for scale up and scale down, respectively. In Algorithm 2 Line 1, the services scale up candidate list is obtained by following the criteria: i) the

Algorithm 2 ScaleUp()

```
1: Get service scale up candidate list, $sList$
2: for $sName$ in $sList$ do
3:     if $sName$ in $srvc2rVMMap$ then
4:         Get the running $vmList$ for the sName
5:         Sort $vmList$ based on VM's $cpuLoadRoom$, i.e. $VM.numvCPUs - VM.loadAverage$
6:         for each $VM$ in $vmList$ do
7:             if $numvCPUs > (loadAverage + cpuBuffer)$ then
8:                 Add a service instance to the $VM$
9:                 break
10:            end if
11:        end for
12:        continue
13:    end if
14:    Resume a VM containing the service $sName$
15:    if Resume is successful then
16:        continue
17:    else if a VM is started containing the service $sName$ within the ScaleUpAllowanceTime then
18:        Skip starting a new VM
19:    else
20:        Start a new VM instance containing the service $sName$
21:    end if
22: end for
```

Algorithm 3 ScaleDown()

```
1: Get service instance scale down candidate list, $sList$
2: for $sName$ in $sList$ do
3:     Get the $vmList$ for the service $sName$
4:     for each VM in $vmlist$ do
5:         Stop and remove the idle service $exName$ instances from the VM maintaining minimum number of sName instances
6:     end for
7: end for
8: Get Idle VM scale down candidates list
9: Suspend the idle VMs from the list based on its channel's idle time in the message bus while maintaining minimum number of service instances
```

length of the RabbitMQ queue for a service is greater than a pre-defined threshold, such as 100 or 1000, or ii) the number of consumers (extractors/Software Servers) for the queue is less than the minimum number configured. In Algorithm 3, Line 1 obtains the service instance scale down list by following the criteria: i) idle queues (no data /activity for a configurable amount of time), ii) idle VMs by using the channels idle time taking care of multiple channels on the same VM. Rest of the code in both algorithms are self-explanatory.

2) Implementation: The elasticity module is a stand-alone program written in Python. The statistics obtained from RabbitMQ and Openstack, as well as the scaling actions information, are written into a MongoDB database so these values can later be analyzed/visualized. We run this module as a system service so that the OS ensures it is up and running, thereby addressing the DTS and DAP's performance, robustness, and availability.

III. Extensibility

As the name suggests, a mutt of software, the Brown Dog services, in particular via Software Servers, are designed to use and manage potentially any piece of software to carry out conversions and extractions. In a manner similar to Apple's App Store, Galaxy's Tool Shed [20], or other such repositories for applications[12] [3] we build a Tools Catalog which allows users to add new tools and their capabilities to the two services. As the DAP and DTS can utilize arbitrary code to carry out operations the Tools Catalog doesn't actually store the actual tools (i.e. code, software), but instead references them typically via a URL to a website or source code repository. What is stored within the repository is information needed to both call these referenced tools and to give credit to their creators so as to motivate the addition of new tools. Control of the tool is done through the wrapper scripts mentioned previously (Figure 2 and Figure 3), and is designed to be as simple and straight forward as possible. Credit for the time being is done via providing citation information, such as a relevant paper, or to the software directly[13]. This can be done as simply as providing a Digital Object Identifier (DOI) and will become more and more important as the scientific community moves towards providing as much credit for software and data as it does for articles.

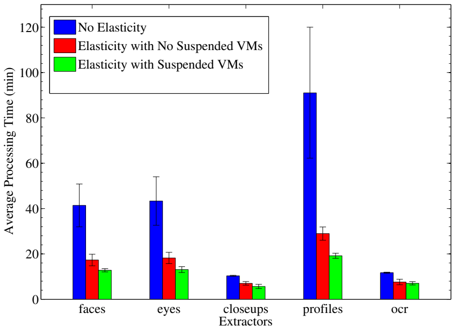


Figure 4: Comparison of average total processing time to serve 1200 job requests by each extractor in Scenarios 1, 2, and 3

IV. Evaluation

With regards to performance of the proposed infrastructure we focus on two measures, scalability, specifically with regards to the very heterogeneous tools that make up the two services, and capabilities, what types of data are we currently able to support. Though the deliberate extensibility of the system plays into the latter criteria it is still not trivial as elements within the system, e.g. the I/O-graph utilized by the DAP, allow for combinations of tools to present additional capabilities (e.g. a chain of conversions to reach a desired target format). Below we describe our results with regards to the elastic scaling of the system as well as a framework we are building towards the continuous evaluation of the capabilities of the system.

A. *Elasticity*

The experimental evaluation is designed to study the effectiveness of our approach and to get insights into the auto scaling behavior of the monitored services. Our experimental setup comprises of:

- Two extractor types - OpenCV based (Open source Computer Vision) and OCR (Optical Character Recognition), both extractors triggered off of images. Specifically we consider four OpenCV extractors to detect faces, eyes, profiles and closeups of faces, and one OCR extractor.

[12] https://www.docker.com/

[13] http://zenodo.org/

- Created two VM images based on Ubuntu Trusty (14.04) with 1 GB RAM, 1vCPU, 10GB disk space for the two extractor types. One image is configured to have one instance of four OpenCV extractors (faces, eyes, profiles, closeups) as services with OpenCV already installed. The second VM image is configured to have one OCR extractor.
- A single instance of the DTS, a RabbitMQ server, OpenStack cloud, and MongoDB.
- 60 test image files in the PNG format with varying sizes between 10KB-4MB. Image contents have a varied number of faces, eyes, closeups, profiles, and text. Each of the images has been tested against the extractors to make sure the extraction process works correctly.
- The queue length threshold is 30 and Idle Time for VM before it could be suspended is fifteen minutes. Minimum number of extractor instances required is two for all extractors.

For our experiments we utilize an OpenStack cloud run out of the NCSA Innovative Systems Lab (ISL). In each test, we uploaded 20 iterations of the 60 images for metadata extraction using the DTS REST API. Each of the five extractors processed all 1200 files.

When scaling up, existing suspended VMs are resumed as opposed to created to respond faster. To evaluate this effect on processing time, we tested three scenarios: without elasticity (denoted as **Scenario 1**), elasticity with no suspended VMs (**Scenario 2**), and elasticity with suspended VMs at the starting of a test (**Scenario 3**). Figure 4 shows the comparison of average total processing time in the three scenarios. The elasticity module reduced the total processing time for all extractors. Among the five extractors, the OCR and closeup recognition extractors take the least CPU time, while the facial profiles extractor is the most CPU-intensive. The reduction rate was 32–68% in Scenario 2, and 45–79% in Scenario 3. The most CPU-intensive extractor, facial profiles, in Scenario 3, had the largest reduction rate, 79%.

The queue length graphs in Figure 5 (top) confirm that with suspended VMs, the system responded to demand surges faster than without, while the bottom graphs show the reason: resuming VMs and adding extractor instances on VMs increased extractor instances faster than starting new VMs. The bottom graphs also show the scaling down behavior.

B. Curation and the CI-BER Testbed

In order to test and report on the application of Brown Dog to archival collections, we attempt to simulate archival processing using the CI-BER[14] [7] collection housed at the UMD Digital Curation Innovation Center. CI-BER contains 52 terabytes, 72 million objects spanning a wide variety of file formats, scientific datasets, and organizational records. Through scripted interactions with the services we gather metrics and identify gaps in function. The scripts are run over and over again and seeded with different sample data files. The technology we use for simulation is a testing framework called Gatling.io[15]. Gatling provides a domain-specific language (DSL) for constructing web-based tests. The test scenarios may be run as single user tests or they may be run in parallel as realistic load tests, simulating the same scenario for hundreds of users at the same time. Gatling produces detailed metrics on service performance under load, which can be used by developers to isolate issues. We write our simulation scripts in the Gatling DSL, seeding simulations with a randomized set of sample files. After each simulation runs we gather up results from Gatling log files and other sources to build a database in MongoDB of simulation results.

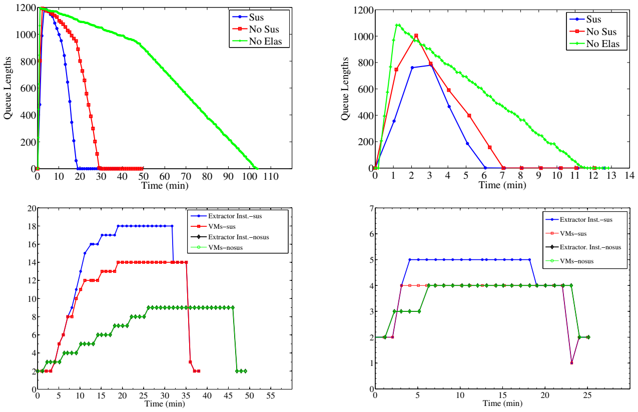


Figure 5: Comparison of queue lengths (**top**) and the number of extractor instances and number of VMs (**bottom**) for the facial profiles (**left**) and OCR (**right**) extractors, with Scenarios 1, 2 and 3.

1) Archival Processing Simulations: The initial approach was to employ Brown Dog for data format conversions typical of digital preservation systems, namely the conversion of legacy materials into current file formats for preservation and access. Twice a day we take a random sample of modern and legacy office files from the CI-BER collection, attempting to convert them all into PDF using the DAP. Any office documents that have no conversion path are flagged as a problem (this can change over time as this is an elastic cloud based system). We periodically analyze missing conversion paths and make these into feature requests for the Brown Dog development team. The Gatling test framework records precise timestamps at various points in the HTTP interaction. For instance we can measure conversion time as the time between when the last byte of the HTTP request was sent, until the time the first byte of the HTTP response is received. Figure 6 shows the performance of the office document conversion over time for 100 file conversions. Each simulation consists of either 100 or 1000 users, each running one conversion. The formats used included: DOC, DOCX, ODF, RTF, WPD, WP, LWP, and WSD. We perform a similar file conversion test for image file formats. Formats currently under test include: TARGA, PICT, WMF, BMP, PSD, TGA, PCT, EPS, MACPAINT, MSP, and PCX. These are converted to the TIFF format and any exceptions are reported to the development team.

While TIFF and PDF are used as preservation formats by

[14] https://ciber.umd.edu

[15] http://gatling.io/

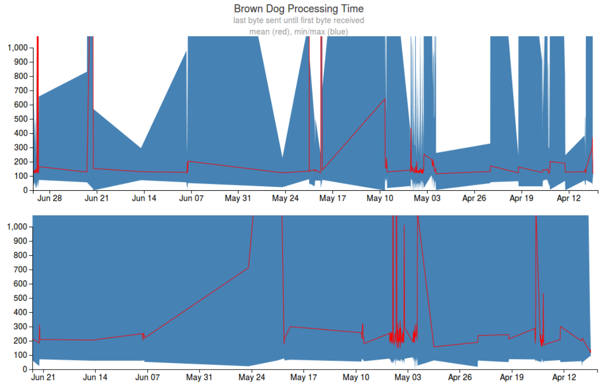


Figure 6: **Top:** Conversion simulation results for document conversions to PDF. **Bottom:** Conversion simulation results for image conversions to TIFF.

some archives, we do not assume that TIFF and PDF are the most desirable formats for all archives. Instead, the TIFF and PDF conversions ensure that the content is not trapped in the original format, that Brown Dog can open the original file and get content out of it. Thus far we have identified numerous samples that are trapped in WordPerfect (WP and WPD), Photoshop (PSD), and Windows MetaFile (WMF) formats. A more robust simulation is also being developed through a policy-based format migration which will sample files randomly from the CI-BER collection and consult a lookup table to obtain the preferred preservation format. The simulation will report missing migration paths, as well as missing migration policies, i.e. data files or formats for which no preservation format has been recommended.

V. Conclusion

We have deployed an alpha release of the two services and begun incorporating tools in support of a number of our use cases (e.g. supporting ecological model conversion via PEcAn, supporting Lidar analysis for our hydrology use case, supporting human preference modeling for our green infrastructure use case, as well as other capabilities suited for more general usage). Further, towards supporting the wide range of users across our use cases we have begun developing a number of client interfaces[16] to leverage the DAP and DTS. These include language specific libraries, a bookmarklet interface that can be used to call the services on arbitrary web pages, a Google Chrome extension, a command line interface, incorporation into a scientific workflow system [21], and incorporation back into Clowder as an example within a content management system. Efforts moving forward aim to add additional cloud infrastructure support to the elasticity module, refine the level of granularity considered during scaling by deploying Docker instances of converters/extractors within a single VM, exploring how we might optimize data movement so as to as efficiently as possible handle large data collections, and incorporating/using additional information relevant to provenance such as the estimates of information loss incurred during specific conversions described in [5].

[16] http://browndog.ncsa.illinois.edu/blog.html

Acknowledgment

This research & development has been funded through National Science Foundation Cooperative Agreement ACI-1261582.

References

[1] N. webmaster, "Vision for cyberinfrastructure framework for 21st century science and engineering," 2013. [Online]. Available: http://www.nsf.gov/od/oci/cif21/CIF21Vision2012current.pdf

[2] E. Deelman *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming Journal*, 2005.

[3] G. Klimeck *et al.*, "nanohub.org: Advancing education and research in nanotechnology," *Computing in Science and Engineering*, 2008.

[4] L. Marini *et al.*, "Medici: A scalable multimedia environment for research," *The Microsoft e-Science Workshop*, 2010.

[5] K. McHenry, R. Kooper, and P. Bajcsy, "Towards a universal, quantifiable, and scalable file format converter," *The IEEE Conference on e-Science*, 2009.

[6] F. Soper, "The pronom file format registry," *Experts Workgroup on the Preservation of Digital Memory*, 2004.

[7] J. Heard and R. Marciano, "A system for scalable visualization of geographic archival records," *IEEE Symposium on Large Data Analysis and Visualization*, 2011.

[8] L. Diesendruck, R. Kooper, L. Marini, and K. McHenry, "Using lucene to index and search the digitized 1940 us census," *Concurrency and Computation: Practice and Experience*, 2014.

[9] M. Lew, "Content-based multimedia information retrieval: State of the art and challenges," *ACM Transactions on Multimedia Computing, Communications, and Applications*, 2006.

[10] D. Garette and E. Klein, "An extensible toolkit for computational semantics," *International Conference on Computational Semantincs*, 2009.

[11] W. Michener *et al.*, "Participatory design of dataone - enabling cyberinfrastruture for the biological and environmental sciences," *Ecological Informatics*, 2012.

[12] J. Myers *et al.*, "Towards sustainable curation and preservation: The sead project's data services approach," *Interoperable Infrastructures for Interdisciplinary Big Data Sciences Workshop, IEEE eScience*, 2015.

[13] M. Dietze, D. LeBauer, and R. Kooper, "On improving the communication between models and data," *Plant, Cell & Environment*, 2012.

[14] K. McHenry *et al.*, "A mosaic of software," *The IEEE International Conference on eScience*, 2011.

[15] A. Rajasekar, M. Wan, W. Schroeder, and R. Moore, "From srb to irods: Policy virtualization using rule-based data grids," 2005.

[16] L. Marini *et al.*, "Versus: A framework for general content-based comparisons," *IEEE eScience*, 2012.

[17] T. Yeh, T. Chang, and R. Miller, "Sikuli: Using gui screenshots for search and automation," *UIST*, 2009.

[18] M. Beckerle and S. Hanson, "Data format description language (dfdl) v1.0 specification," *Open Grid Forum*, 2014.

[19] G. Galante and L. C. E. d. Bona, "A survey on cloud computing elasticity," in *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, ser. UCC '12, 2012, pp. 263–270.

[20] J. Goecks, A. Nekrutenko, and J. Taylor, "Galaxy: A comprehensive approach for supporting accessible, reproducble, and transparent computation research in the life sciences," *Genome Biology*, 2010.

[21] R. Kooper *et al.*, "Cyberintegrator: A highly interactive scientific process management environment to support earth observations," *Geoinformatics Conference*, 2007.