Command and Control Subsystems Report

Jake Vossen: OREPACKAGERS

April 4, 2019

1 Subsystem Description

The Command and Control subsystem is the subsystem responsible for converting the requests that have been collected into downloaded data to be distributed to users. It starts by receiving a list of `request` objects - a structure for containing information about each request in Python. To prevent confusion, the mono-spaced `request` will refer to they Python object itself, whereas plain "request" refers to the concept of a user request.
With this list of requests, the first thing it does is use the Python `multiprocessing`[1] library to split work up between the different threads on the computer. While this software is designed for low end machines to be more accessible to developing areas, most computers[2] in recent times will have more than 1 CPU core (including the Raspberry Pi[3]). This allows for the processor to split up all the requests, and execute them in parallel, instead of waiting for each one to finish individually, which can provide an decrease in time spent downloading files, as explored in section 4.2.
When downloading a `request`, it determines the type of request. The types are URL, search, YouTube, and ipfs. Figure 1 is the flow chart to determine the method used for downloading an asset.

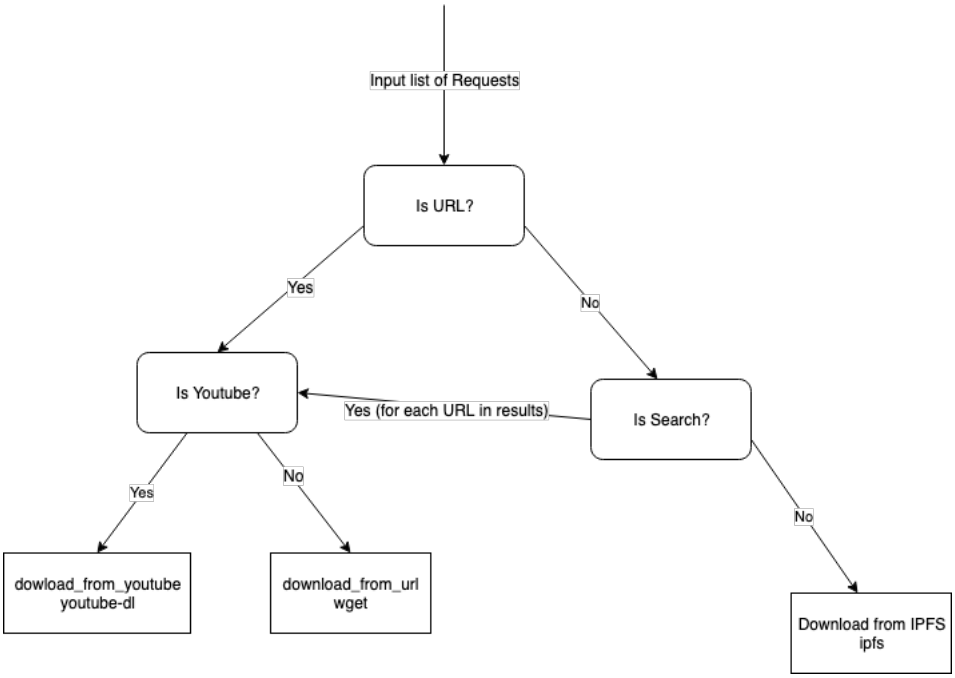

Figure 1: Diagram describing flow chart of downloading each `request`. Jake Vossen, 2019-04-01

The steps for each type of request is outlined below.

1.1 URLs

URLs are your basic websites, such as `https://en.wikipedia.org/wiki/Monty_Python_and_the_Holy_Grail`, or `https://www.nytimes.com/2019/03/27/technology/turing-award-ai.html`. This is for users who already know the content they want. In the backend, the Python program is going to use the `wget` utility. Specifically,

`wget -E -H -k -K -p -P path url robots=off` where `path` is the output directory and `url` is the url that has been requested. Each of those letters does something to modify the request to created the desired behavior. For example, `-p` tells `wget` to download pictures and things linked on the website so the page appears exactly as it would in a web browser. More details for each option can be found in the `wget` documentation[4]. All of those options ensures that downloading the URL requested gets the website exactly as it appears in a browser, including linked images. Additionally, it works with PDF and ZIP files, which is really important to ensure all possible media can be obtained. This method is also used by other parts of the program.

1.2 Search

Sometimes the user will not know exactly what they want, so functionally was added to get the first page of Google search results (top 10 results). The `googlesearch`[5] library was very helpful for this. This library provides a list of the URL's on the first page of a Google Search. With that list, the established URL method described in section 1.1 is used to download each URL (it also checks to ensure to use the YouTube method if it is a YouTube link). The results are each in their own folder named based on the Google search rank (1 is first result, 2 is second result, etc).

1.3 YouTube

It is well known that lot of quality educational and entertainment content is in video format, and the majority of that content is on YouTube. That is why the users can request YouTube videos (through a link, or a result from the search function). In this case, the `youtube-dl`[6] program allows for content retrieval. A user can either request a YouTube video directly, or use the URL method which will detect the YouTube link and use this method.

1.4 IPFS

IPFS stands for "InterPlanetary File System", which is a "A peer-to-peer hypermedia protocol to make the web faster, safer, and more open"[7]. The internet that is familiar to most people is the client-server model[8], but IPFS changes that so everyone is both a client and a server. Media is distributed based of the crytographic hash, a unique ID for each object, instead of a URL. Anybody can add objects, and when requesting an object, it can be downloaded from any number of servers, not just the original person hosting the server. The `ipfs` command line utility[9] is used to retrieve objects. This greatly expands the functionality of our product, because the operator can now do some content retrieval with no access to the internet. If two of these backpacks meet where there was no internet, they can easily exchange content on the IPFS, which could increase speed and decrease the cost of retrieval for those requests.

2 Interfaces with Other Subsystems

Software is all about abstraction, so it is important to clearly define where the Command and Control subsystem will interface with the other subsystems. Ideally, all the other subsystems will work independently and a couple of links will get everything working together. These links are shown in Figure 2, and explained in further detail below. The interfaces are left inside the Python code in the appendix of this document, just commented out with the `#` character.
Most of the meshing is done through the `request` object, which can be found in the appendix.

2.1 Input - List of Requests

Python, like all programming languages, only stores objects in memory while the program is running. That means when the program is shut down (or the machine is powered off), the data generated must be saved somewhere on the device or else that information would be lost. In this case, what is important is to be able to store the `request` objects. This is completed by the data management subsystem. This means my subsystem will call `get_all_requests()` which will retrieve the data from the hard drive of the machine about the requests, create the objects again (as they where destroyed from memory when the program shut down), and return that information to Command and Control.

2.2 Output - Download path and status

Once my subsystem completes it's download, it needs to update the database about the new status. This is again through the Data Management subsystem. To ensure each object is downloaded only once, each `request` has two properties: `file_location` and `downloaded_status`. Once Command and Control has completed a download, it will call the Data Management method `update_request(r)` where `r` include the changes to `file_location` and `downloaded_status`. So the Data Management subsystem knows which database entry to update, a Universally Unique Identifier (UUID)[10] is used to identify each request.

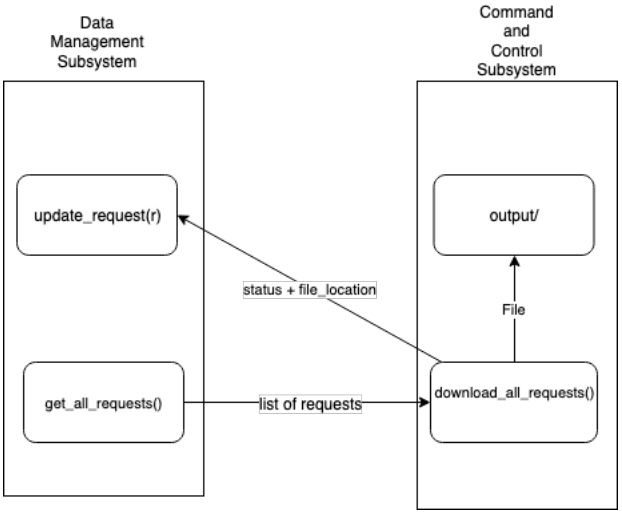

Figure 2: Diagram describing the inputs and outputs of the Command and Control sub-system `request`. Jake Vossen, 2019-04-01

3 Stakeholder Considerations

The primary concern for the Command and Control subsystem is ensuring that the primary forms of retrieval are supported. In an interview with Anthony Wanjiru, who lived in Kenya for 35 years, he said the most important types of content are either a URL (either as a web page, pdf, etc), YouTube, Google Searches, and while IPFS isn't widely used, it has a lot of potential[13]. He also said that local content is on the rise, and a method to upload content would be really beneficial. While this was not implemented at this time, due to specifics with integration with the other subsystems, this could be implemented later.

3.1 Pricing

Due to the emphasis of using our system in low income areas, all software that is used in this process is free to use and distribute. This optimizes the number of people that can use this software.

4 Validation

4.1 File Retrieval

Testing is one of the most fundamentally important things in software development. It is easy to write code, but it is not useful if it can't work under a variety of difficult tests. A handful of curated tested designed to test the boundaries of the code have been picked. To verify the results, there are two primary methods. The first, is a SHA-512 sum, which is a crytographically secure way to ensure two files are the same[11], the first file being originally from a manual download, and the second one from the Command and Control Subsystem Python program. If the two hashes are the same, the subsystem was successful. However, this cannot be used for all tests, because when downloading from a website, the original HTML files are modified for browsing offline (see the `-k` option on `wget`). This makes it impossible to cryptographically ensure that the requested file is the same as the downloaded file, so a visual analysis is preformed to analyze the two documents and ensure that they are identical. The table below explains the tests preformed, where the "Analysis Method" is either visual or hash to preform the check.

Table 1: File Download Tests Preformed, Jake Vossen, 2019-04-01

Type	Value	Expected Result	Analysis Method	Expected = Actual?
URL	https://www.gutenberg.org/cache/epub/2265/pg2265.txt	Text of "Hamlet" By Shakespeare	Hash	Yes
URL	https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/46507.pdf	PDF of Google Machine Learning Research Paper	Hash	Yes
URL	https://en.wikipedia.org/wiki/Monty_Python_and_the_Holy_Grail	Wikipedia entry for "Monty Python and the Holy Grail"	Visual	Yes
YouTube	https://www.youtube.com/watch?v=Gbtulv0mnlU	NASA Shuttle Recovery Video	Visual	Yes
search	What Is the Airspeed Velocity of an Unladen Swallow?	Google search results	Visual	Yes
ipfs	/ipfs/QmS4ustL54uo8FzR9455qaxZwuMiUhyvMcX9Ba8nUH4uVv/readme	IPFS introduction document	Hash	Yes
ipfs	/ipfs/QmVLTMHtLRhnft3QspDx4qTJeXY6hiib1j77UfQmY54CGe/mosaic.png	300 MB picture of the moon	Hash	Yes

4.2 Multi threading

The other important validation is to ensure that the multi threading described in section 1 improves results. I ran the program with 8 fairly large requests and used the `time`[12] utility to measure the run time of the program. The results are shown below:

Table 2: Tests Preformed, Jake Vossen, 2019-04-01

1 Core	2 Cores	4 Cores	8 Cores
161.277 seconds	53.927 seconds	45.984 seconds	46.448 seconds

From this data, it is clear that the multi threading is successful in decreasing the time required to download multiple requests. Further work could be done to multi thread the `googlesearch` library, as the current multi threading solution does not support sub threads.

5 Idea Generation / Software Choices and Requirements

Below is the reasoning for each software library and tool picked. It also acts as a requirements list

- Python - Easy for beginners to understand, has a lot of libraries, powerful
- `googlesearch` - It is really difficult to parse the URLs and submit a search to Google through plain HTTP requests, and this library takes care of this as well as providing updates if the format changes in the future.
- `youtube_dl` - Written in Python, does exactly what is needed, and also supports sites that aren't YouTube which expands the usefullness of this project
- `wget` - Open source, widely used, and can create a snapshot as the website exactly appears, instead of just the file at the specific URL.
- IPFS - Allows for the decentralization of content and therefor increases download speed

6 Appendix

6.1 Command and Control Python Program

```
from request import Request
from user import user
from datetime import datetime
import subprocess
import os
from googlesearch import search
import youtube_dl
import multiprocessing
from multiprocessing import Pool

def download_all_requests(requests):
    threads_count = multiprocessing.cpu_count() # use all of the threads on the
        system that we can
    # Below are the hard coded values for the testing of the effectiveness of
        threads
    # threads_count = 1
```

```
    # threads_count = 2
    # threads_count = 4
    # threads_count = 8
    with Pool(threads_count) as p:
        p.map(download_request, requests)
    # The above code is the same as the code below, above will do it with as many
        threads as possible
    # for r in requests:
        # download_request(r)

def mkdir(path): #mkdir = Make Director, creates a folder if it does not exist
    if not os.path.exists(path):
        os.makedirs(path)

def download_request(r):
    if r.kind == "URL": # we are dealing with a plain old HTTP request
        url = r.value
        path = "output/" + r.uuid
        download_from_url(url, path)
    if r.kind == "search":
        search_list = list(search(r.value, stop = 10)) # call the google_search
            library to get the list of urls for the search terms stored in r.value
        for i in range(len(search_list)): # it would nice to multithread this, but
            we are already multithreaded at the request level. More work could be
            done to improve this system.
            url = search_list[i]
            path = 'output/' + r.uuid + '/' + str(i + 1) + '/'
            if ('youtube' not in url): # Youtube vidoes are not going to be able to
                work, those will be for the youtube request
                download_from_url(url, path)
            else:
                download_from_youtube(url, path)
    if r.kind == "youtube":
        path = "output/" + r.uuid + '/'
        download_from_youtube(r.value, path)
    if r.kind == "ipfs":
        path = "output/" + r.uuid + '/'
        download_from_ipfs(r.value, path)

    # mark_as_downloaded(p) # this is where this function will intergrate with data
        managment to update the file location and downlaoded status

def download_from_url(url, path):
    mkdir(path)
    subprocess.call(r'wget -E -H -k -K -p -P ' + path + ' ' +  url + ' robots=off ',
        shell=True)

def download_from_youtube(url, path):
    mkdir(path)
    ydl_opts = {
        'outtmpl': path + '%(title)s' # add title to file name
    }
    with youtube_dl.YoutubeDL(ydl_opts) as ydl:
        ydl.download([url])

def download_from_ipfs(ipfs_hash, path):
    mkdir(path)
```

```
    subprocess.call(r'ipfs get ' + ipfs_hash + ' -o ' + path, shell=True) #this
        requires the IPFS daemon running

def main():
    #requests = get_all_requests() # This is where command and control will
        interafce with Data Managment. It will return the list of requests from the
        database.
    requests = [] # Because we can't do the above method, we are going to create a
        mock list with requests
    # tests
    requests.append(Request("URL", "https://www.gutenberg.org/cache/epub/2265/pg2265
        .txt", user("Jake", "Vossen", "jakevossen", "asdf"), datetime.now()))
    requests.append(Request("URL", "https://static.googleusercontent.com/media/
        research.google.com/en//pubs/archive/46507.pdf", user("Jake", "Vossen", "
        jakevossen", "asdf"), datetime.now()))
    requests.append(Request("URL", "https://en.wikipedia.org/wiki/
        Monty_Python_and_the_Holy_Grail", user("Jake", "Vossen", "jakevossen", "asdf
        "), datetime.now()))
    requests.append(Request("search", "What Is the Airspeed Velocity of an Unladen
        Swallow?", user("Jake", "Vossen", "jakevossen", "asdf"), datetime.now()))
    requests.append(Request("search", "Library of Congress", user("Jake", "Vossen",
        "jakevossen", "asdf"), datetime.now()))
    requests.append(Request("youtube", "https://www.youtube.com/watch?v=NtrVwX1ncqk"
        , user("Jake", "Vossen", "jakevossen", "asdf"), datetime.now()))
    requests.append(Request("youtube", "https://www.youtube.com/watch?v=Gbtulv0mnlU"
        , user("Jake", "Vossen", "jakevossen", "asdf"), datetime.now()))
    requests.append(Request("ipfs", "/ipfs/
        QmS4ustL54uo8FzR9455qaxZwuMiUhyvMcX9Ba8nUH4uVv/readme", user("Jake", "Vossen
        ", "jakevossen", "asdf"), datetime.now()))
    requests.append(Request("ipfs", "/ipfs/
        QmVLTMHtLRhnft3QspDx4qTJeXY6hiib1j77UfQmY54CGe/mosaic.png", user("Jake", "
        Vossen", "jakevossen", "asdf"), datetime.now()))

    download_all_requests(requests)

main() # start the program
```

6.2 Request Object

```
import uuid
class Request:
    'This is the main object that stores information about somones request'
    def __init__(self, kind, value, user, date):
        self.kind = kind # options: "search, url, IPFS hash, git URL
        self.value = value # the value associated with the type
        self.user = user # The user associated with this request. We can make a user
            object later if we feel like it
        self.date = date # This should be a date-time object
        self.downloaded_status = False # This is not included in the constructor
            because state should always be false when starting
        self.uuid=str(uuid.uuid4())
        self.file_location = "" # There is no downloaded location when the object is
            created
    def set_downloaded_status(self, state):
        self.downloaded_status = state # Should be a boolean
    def set_file_location(self, loc):
```

```
        self.file_location = loc # Is there a file object that would be better to
            use than a string?
```

7 Bibliography

[1] "multiprocessing - Process-based parallelism," *multiprocessing - Process-based parallelism - Python 3.7.3 documentation*. [Online]. Available: `https://docs.python.org/3.7/library/multiprocessing.html`. [Accessed: 03-Apr-2019].

[2] "Hardware Survey - CPU Cores," *Hardware Survey - CPU Cores*. [Online]. Available: `https://www.pcbenchmarks.net/number-of-cpu-cores.html`. [Accessed: 03-Apr-2019].

[3] "Raspberry Pi 3 Model B," Raspberry Pi. [Online]. Available: `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/`. [Accessed: 03-Apr-2019].

[4] "GNU Wget 1.20 Manual," *GNU Wget 1.20 Manual*. [Online]. Available: `https://www.gnu.org/software/wget/manual/wget.html`. [Accessed: 03-Apr-2019].

[5] MarioVilas, "MarioVilas/googlesearch," GitHub, 06-Mar-2019. [Online]. Available: `https://github.com/MarioVilas/googlesearch`. [Accessed: 03-Apr-2019].

[6] Ytdl-Org, "ytdl-org/youtube-dl" GitHub, 03-Apr-2019. [Online]. Available: `https://github.com/ytdl-org/youtube-dl/`. [Accessed: 03-Apr-2019].

[7] "IPFS is the Distributed Web" IPFS. [Online]. Available: `https://ipfs.io/`. [Accessed: 03-Apr-2019].

[8] "Client-Server Mode," [Online]. Available: `https://web.cs.wpi.edu/~cs513/s07/week1-unixsock.pdf`. [Accessed: 02-Apr-2019].

[9] "IPFS Documentation," *Install IPFS – IPFS Documentation*. [Online]. Available: `https://docs.ipfs.io/introduction/install/`. [Accessed: 03-Apr-2019].

[10] "A Universally Unique IDentifier (UUID) URN Namespace," *IETF Tools*. [Online]. Available: `https://tools.ietf.org/html/rfc4122.html`. [Accessed: 03-Apr-2019].

[11] C. H. Romie, "Secure Hash Standard (SHS)" Aug-2015. [Online]. Available: `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf`. [Accessed: 02-Apr-2019].

[12] "time," *time(1) - OpenBSD manual pages*. [Online]. Available: https://man.openbsd.org/time. [Accessed: 03-Apr-2019].

[13] J. J. Vossen and A. Wanjiru, "Discussions of Content Distribution in Kenya," 13-Feb-2019.