Thesis (UAS)

Degree Programme in Information Technology

Internet Technology

2015

Deepak Panta

WEB CRAWLING AND SCRAPING

–developing a sale-based website

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS (UAS) │ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelor of Engineering | Information Techonogy

2015 | Number of pages 41

Instructor:Ferm Tiina

Deepak Panta

WEB CRAWLING AND SCRAPING

-developing a sale-based website

Most customers spend a fair amount of time searching various items that are on discount. Retailers try to attract as many customers as possible with discounts. While there are ample web sites that offer the service of providing information of discount on items such as clothes, electronics or household appliances, there hardly are websites that offer this service on grocery items.

This thesis describes how Scrapy is used to crawl and scrape discount information from major Finnish grocery stores and display the information in a web site using Django as framework. Since the items are on discount for limited period of time the crawlers and scrapers are scheduled to run on pre-defined intervals. Once the new items are available for scraping the old information is removed and updated with new information.

This thesis also explores the various aspects of the tools (Django, Scrapy, Dynamic Django Scraper, Celery) used to build the website. The outcome of the thesis is a fully functional sales-based web site that displays price, image and additional information about items along with the location of nearby markets in grid format.

KEYWORDS:

Web scrapers, Web crawlers, Bots, Django, Celery, Selenium, Dynamic Django, Scraper, Scrapy, Python

TABLE OF CONTENTS

FIGURES

LIST OF ABBREVIATIONS (OR) SYMBOLS

XML	XML stands for Extensible Markup Language that is used to describle data .
HTML	HyperText Markup Language which is used to create web pages.
XPath	XPath is syntax to describe XML objects and is also used to navigate through XML nodes
Django	A popular web framework
Scrapy	Python based web crawling and scraping tool
DDS	Dynamic Django Scraper which is Scrapy based on Django
Celery	Python based asynchronous task scheduler
Selenium	A brower automation framework used to test softwares and web applications
AMQP	AMQP stands for Advanced Message Queueing Protocl capable of queueing and routing message

1 INTRODUCTION

With the advent of internet and its easy access, people's buying habits are constantly changing. Consumers are using websites that help them find cheaper flights, hotels and discount vouchers more than ever before. According to a survey by bestvaluefares.co.uk[1], nine out of ten adults regularly use online vouchers, tokens and offers to get the best deals on travel, tourism and entertainment

Since consumers are making decisions based upon discounts retailers are coming up with more and more offers and discounts. Discounts are the ways of luring customers into buying more items. Several items are bought not out of necessity but because of the huge amount of discount retailers offer. Consumers switch between tabs in browsers to look for discounts offered by retailers. The most convenient way is to view all the items on a single web page. There are websites that aggregate discounts and offers such as Groupon, my Voucher Codes, Voucher Seeker etc.

When it comes to food items, there hardly are any sites that offer aggregation of discount information. Customers have to visit the retailer's web pages for this information. It is quite frustrating when consumers have to visit each retailer's website. The purpose of this thesis is to reduce this problem to certain extent by making a website that informs consumers about discounts and sale offers provided by the retailers especially on grocery items. This thesis uses web scraping and web crawling techniques to aggregate the discount and offers data available in Finnish grocery stores into a single web site.

The thesis begins with the introduction chapter which states the purpose of the thesis. Chapter 2 explains web crawling and web scraping. Tools and techniques used for crawling and scraping are introduced under this chapter.Tools that are required to develop the web site for thesis are presented in Chapter 3. A basic introduction and functions of the used softwares/tools along with their advantages and disadvantages are also discussed. Chapter 4 covers the implementation of the mentioned tools. Chapter 5 describes the result of thesis along with future improvements and advanced features that can be implemented in the website . Finally, the thesis is concluded in Chapter 6.

2 BACKGROUND

Web crawling and web scraping are the common techniques in modern information technology. Under the hood of search engines are the web robots. Web scraping is the technique of processing the documents in the web, extracting various kinds of data and saving them to local database for later reference. Similarly, the process of systematically browsing web by web robots is called web crawling. When enter key is pressed with search keywords, user is presented with lists of results. Search engines use web crawling and scraping techniques to provide this service.

Indexes are the systematically arranged keywords that allow to locate the information in a record. The process of creating such index is indexing. Web crawlers which are also referred to as web robots, an ant, an automatic indexer or spider, are the programs that browse the web in an automated way to create a web index. Web crawlers are in particular used by search engines and web sites whose content should be filled with third party's websites.

Web scraping is also commonly referred to as web harvesting or data extraction. Although it is used interchangeably with the term web crawling, there lies a marked difference between them. Web crawling is the process of crawling web pages in order to make an index while web scraping is the process of saving anything for later use, even a simple copy and paste is a simple form of web scraping. Since web pages have to be followed web crawlers need the use of internet which is not required for web scrapers.

2.1 Web Crawling

A web crawler is an internet bot that systematically browses the World Wide Web, typically for the purpose of web indexing.[2]. The process of crawling starts with a list of web pages that are to be visited, also called seeds which consists of popular web pages from the server that gets millions of hits. The crawler visits each site mentioned in seeds in an automated way and marks any hyperlinks that appear to another list called frontier. The frontier is crowded with the list of URLs that are to be visited once the seeds are finished. The crawled web pages are stored in local database as snapshot and can behave as if the pages are on the live web.

Crawlers are widely used by search engines like Google, Yahoo or Bing. Google is the first company that published its web crawler which has two programs, spider and mite. The first one maintains the seeds while the later is responsible for downloading web pages.

Googlebot and Bingbot are the most popular spiders owned by Google and Bing respectively. The web crawlers are capable of finding the frequently used words to make an index. So whenever a user enters some search text, it is compared with the words in the database to yield the results that have desired words in the page content. When the search words are located in multiple links the links that are most popular are displayed in general according to algorithm.

One anonymous web administrator claims that 44.4 GB of bandwidth was consumed in one month by crawlers on his website [3]. Since crawling consumes lot of bandwidth and CPU process for both server being crawled and the crawling host, it is always advisable to follow certain rules or policies. Listed below are some crawling policies that will make the crawling process much more efficient and reasonable.

2.1.1 Selection Policy

As of 2013 there are 2,756,198,420 internet users along with 672,985,183 websites [4]. It is easy not to know what is important and what is not with all these huge information. As already mentioned, crawling is a resource intensive process one must know what to crawl and where to. Crawlers download only a small percentage of information from pages so it is important that the downloaded part is relevant and not just some random part. Generally crawling should start in those servers which get millions of hits per day or those links that are visited most or those that appear in most of the sites. In order to achieve the good selection policy following points should be addressed.

1. Ignoring links that are already crawled

The same link might appear in thousands of web pages. When a crawler crawls the pages it is important to know if the page has been downloaded already or not. Downloading the same page again and again is a waste of bandwidth and storage size.

2. Path-ascending crawling

Sometimes it is difficult to find information if the pages are not mentioned or linked anywhere. Path-ascending crawling is the process of ascending through the URL path. For example given a seed path of http://turkuamk.fi/turku/ict/blank.html path ascending crawling will crawl /turku, /ict and /turku/ict. Its usefulness lies in locating isolated information not linked anywhere.

3. Focused Crawling

Focused crawling also called topical crawling is the process of downloading the pages that meets the certain prerequisite such as topic or title. It is a useful technique to maximize the effort of getting the required resources since the area where the crawling needs to be done is narrowed a lot.

2.1.2 Politeness Policy

Crawling requires a huge amount of bandwidth and CPU overload on the server being hit. If there are lots of crawler requesting pages or downloading files it can severely impact the performance of the server, sometimes even disrupting the network and servers. Therefore crawlers need the delay interval between two successive requests. Generally this delay depends upon the crawlers but a delay of 10 to 20 seconds is considered acceptable. Too long delay, on the other hand, is not acceptable as it might take days for crawlers to download the required information. Server administrators can use robot exclusion protocol to address the problems that can be created by bad crawlers.

Robot Exclusion Protocol

Crawlers come with various costs such as large bandwidth, server overload, network disruption and crashing of routers. Server administrators have a tool called robots exclusion protocol to mitigate these problems. It is a protocol which has a definition of which part of the server can be accessed by crawlers and which are denied. It is generally referred to as robots.txt protocol. All crawlers before downloading any pages read this file to know what is allowed and what is restricted to download. The disadvantage of this protocol is that the contents of the web are made public. When certain information is placed under disallow directives, crawlers become aware of the existence of this information. Malicious computer users can use this to benefit themselves or disrupt the service. Robots.txt file has two fields allow and disallow which advises the crawlers about what can be crawled and what is forbidden respectively.

Server administrators usually put a text file named robots.txt at the root of the web site. This file contains various tags or filters to either allow or disallow the crawlers to crawl the site. Some of the common excluding or permitting examples are:

Disallow:/var/

The above robot.txt informs crawlers not to crawl the /var directory. It can specify which browsers are allowed or which crawler are permitted to download files too.

User-agent: GoogleBot

Disallow: /

This informs google crawler that the root directory cannot be crawled while other bots can crawl the root directory.

2.1.3 Revisit Policy

Crawlers are mostly used by search engines for the purpose of indexing the pages. Thousands of sites are added to web everyday while other thousands are deleted. Information is continuously added, deleted or amended. It is absolutely necessary for search engines to keep up to date information. Therefore, pages have to be visited frequently to update with the changes.

Crawlers have a predefined policy on when and what pages to revisit. Search engines use two elements freshness and age as functions. Freshness is binary expression to indicate if the downloaded information is still accurate or not, while age describes how outdated the downloaded information is. They are expressed as -

Freshness:

$$F_p(t) = \begin{cases} 1 & \text{if } p \text{ is equal to the local copy at time } t \\ 0 & \text{otherwise} \end{cases}$$

Figure 1. Equation relating to freshness of page [5]

Age:

$$A_p(t) = \begin{cases} 0 & \text{if } p \text{ is not modified at time } t \\ t - \text{modification time of } p & \text{otherwise} \end{cases}$$

Figure 2. Equation relating to age of page [5]

Cho and Gracia-Molina, studied two re-visiting policies of web crawlers.

1. Uniform Policy: All pages are re-visited at the same frequency irrespective of the changes in the content
2. Proportional Policy: Web pages that are subject to frequent changes are visited more often

Cho and Gracia-Molina concluded that the uniform policy is more desirable in terms of freshness of page compared to the proportional policy in both real world and simulation. However, in real life search engines tend to ignore pages that changes too frequently.

2.1.4 Parallelization Policy

The process of downloading is fairly lengthy process if few crawlers are deployed. However if many crawlers are running on the same machine it will slow down the performance. Under the parallelization policy many computers are run at the same time to index the web thus by creating load balancing between crawlers. Since the crawlers are disturbed among different machines the failure of single computer doesn't affect the other crawlers. There are two types of parallelization policy.

1. Dynamic assignment: Using this technique one central server assigns the URLs for crawlers dynamically.

2. Static assignment: The URLs to crawl are defined at the beginning with certain fixed rule.

2.1.5 Crawling Strategies

Different algorithms are used when it comes to crawling but there are two main techniques. Since the web changes constantly it's important to crawl the relevant items before unwanted pages are crawled to save the bandwidth and resources. It has always been the topic of how to make crawlers more effective. These two algorithms are mainly followed by search engines.

1. Breadth First Search

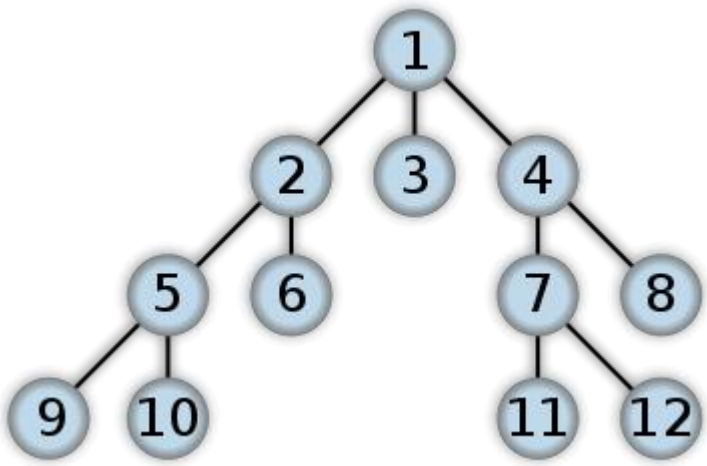

Figure 3. Graph tree [12]

It is a technique where all the links in a page are followed in sequential order before the crawler follows the child links. Here graph number 1 is the web page that has three links (2, 3 and 4). Page 2 has two links in it (5 and 6), similarly page 4 also has two links (7 and 8).

Under BFS algorithm first of all links 2, 3 and 4 are visited. Then the child links 5 and 6 are crawled followed by 7 and 8 respectively. This process continues until the crawler reaches the end of tree if searched item is not found.

Since the child links can only be generated from the parent links, crawlers need to save all the parent links in a page in order to follow the child links. This process consumes lot of memory and computer will run out of it after few crawling. Similarly, if the searched item belongs to deeper child link BFS takes much more time to find it. However BFS is effective when finding more than one search item if there exist many and getting caught in loopback is never a problem for BFS.

2. Depth First Search

DFS is an algorithm where the crawler starts with the parent link and crawls the child link until it reaches the end and then continues with another parent link. For example in Figure 3, if DFS is implemented, the crawler begins at 1 then to 2. Once the crawler reaches link 2, it continues with link 5 followed by link 9 and 10 and then jumps back to link 6.

The advantage of DFS comes in terms of memory consumption. Since it does not have to save all the parents links in a page, it consumes relatively less memory than BFS. While this may consume less memory, DFS is vulnerable to being lost without finding the required search item. With this algorithm, one may or may not find the solution.

2.1.6 Challenges for web crawlers

1. Deep web

The content of the web that is not visible via traditional search engines or the content of web that are not indexed by search engines is called deep web. The Internet is a huge ocean of information and only a part of it has been indexed, there still lies lots of web pages that has not been accounted for. Since deep web is hidden, it is hard to estimate the size of it . However, scientists suggest that the deep web is 4000 to 5000 times bigger than the web we are used to. There has been great improvement on how to crawl the deep web but still it is challenging. The reason why it is hard to crawl deep web is because of its independency. Most of the contents of the deep web are standalone pages. The pages are not linked to one another and there is no way to know these pages via other pages. Crawlers follow links and download them so it is hard in cases like these.

Similarly, most of the pages in deep web require some sort of authentication, registration or logging in. Web pages are available only once the form is filled. Traditional crawlers have no idea of forms and how to fill them. So the information behind these forms is unaccounted. In recent developments, programmers are developing crawlers that are able to identify forms and able to fill them with required values. However, the problem is that crawlers do not know if the information behind those forms is informational or not.

2. Multimedia

The Internet is composed of images, videos, texts and lots of other multimedia files. Although it is easy for search engines to crawl the text, it is hard for multimedia documents like image or video. Sometimes even the text information is found in images and crawlers can not read those texts from the image. It is hard for search engines to return a search results on multimedia if no description on text format are available. For example if someone searches for “a bird sitting on tree” search engines have no idea which image to present if nothing is mentioned in the description of the image or if it does not have any tags. Similar is the case with videos.

3. JavaScript rich sites

Web pages are becoming more and more user-friendly and responsive due to the development in the field of web. Although JavaScript makes web pages more interactive, they create lots of obstacles for crawlers. There have been tools and techniques to crawl AJAX enabled pages, however these techniques need more resource and more coding. Selenium is an automation testing software for web applications which can be used to crawl and extract data from AJAX-enabled sites.

2.2 Web Scraping

The process of simulating the human exploration of websites to extract information is called web scraping. Even though it is closely related to web crawling, it is a different technique. The main purpose of web scraper is to convert unstructured data found on the internet to structured format for analyzing or later reference. Web scraping is also needed when web sites that is needed to be saved does not provide RSS feeds or API.

2.2.1 Tools and techniques used for web scraping

With the advancement of software in the web, there are lots of ways to scrape data from the web. There are lots of tools available in different programming languages. Following are some of the ways and tools used in scraping.

1. Parsing the DOM

Document Object Module (DOM) is an API which defines how the elements in the document are accessed and manipulated. The elements in DOM are defined in a systematic way called DOM tree. Whenever a user browses any page in the web, first the computer downloads the page and parses it to DOM tree to render. This DOM can be parsed where the nodes to be scraped can be defined.

2. Software

Several software are available these days that enables users to scrape the web with the minimal coding. A user with a basic understanding of HTML, XML can easily perform the job. These software are graphically user friendly and present lots of tutorials on performing the scraping. Some of the popular software are :

Scraper: It is a google plugin that is simple and user friendly. One can click and scrape data and can easily add to scrape other similar data from the page.

Import.io: This is a browser-based scraper. All a user needs to do is select the data that needs to be scraped.

ScraperWiki: This is a little advanced scraper but more powerful. Users can customize their scraping need and it can scrape data on a huge amount.

Kimono Labs: It is a simple browser based scraper where all a user needs to do is write the URL of the page that needs to be scraped and mention what kind of data to extract and how often.

Scrapy: It is a python based web crawling and scraping framework. This framework will be used in this project for the purpose of scraping and crawling.

3. HTTP Programming

This technique uses the requesting of HTTP to the remote server using socket programming.

4. Human copy and paste

Even advanced software or tools fails sometimes. Sometimes it is the only option available for saving the information.

2.2.2 Legal Issues

The answer of whether scraping is legal or not depends upon what is done with the scraped data. The information on the web is available for public, so anyone can save those data for later use or research. But scraping contents from the web to sell later is crime. Most of the times what can be done and what cannot be is mentioned in the terms of use of the web site.

Violating these mentioned terms of agreement is obviously against the law. One of the famous cases involved Bidder's Edge, an auction aggregator sued by eBay on 10th December 1999 for scraping the contents of eBay and using them for commercial purpose[6]. There is not a clear line that marks the difference in what is ethical and what is not in this debate. The views differ from company to company and web site to web site. However, it's illegal to reproduce the contents available on the other web pages for the commercial uses. Web scraping on the small scale generally for personal use or for research are agreed upon by most companies as legal.

2.2.3 Uses of Scraped data

The information is scraped for different purposes. A common pattern of using the scraped information can be as follows:-

1. Research

Most of the data extracted can be plotted against some input mechanism so as to compare these data. Research needs lots of data, especially data from the past that can be compared to similar data from present.

2. Web site mirroring

Some servers get more hits that they cannot handle and thus crash. One way to mitigate this problem is to have a duplicate site. This can be done using web scraping. A scraper can copy all the contents of a web page to make exact copy of it so that the load can be shared among servers.

3. Price comparison

Scrapers can be used to scrape prices of items and thus can be compared with similar items from different vendors. This service provides users a choice while shopping. Other uses of scraper can be:

Detecting the change in the web pages

Crawling data on forums for post-processing or analyzing

Scrape offers and discounts

Aggregate listing of jobs and vacancies

Data mining

2.2.4 Challenges for web scrapers

Challenges related to web scraping are very similar to the challenges of web crawling. Contents loaded via AJAX which are based on scrolling are quite hard to scrape. Web administrators make it harder on scrapers by modifying the contents of website. For example when the next button is clicked on amazon site via crawlers, the request can be modified to deliver a different next page which is not real. Similarly location based web sites such as www.yhd.com are hard to scrape since the contents of the web page differ based on where the user is accessing it from.

The next chapter describes what tools are used to develop the sale based web site.

3 TOOLS

The sale-based web site is a result of collection of various python packages and software. In order to create this web site Django, Celery, Dynamic Django Scraper, Scrapy and other python packages were used. This chapter describes these packages, their configurations and the reason for their preference over other similar package.

3.1 Scrapy

Scrapy is a Python-based web scraping and crawling framework which was officially released in August 2008. With 1.5k github followers, it is one of the most popular web scraping and crawling tool. It was created at Mydeco, a London based company specialising in the field of web aggregation and e-commerce. In 2011 Mydeco and Scrapinghub (both are web aggregation companies) came together to work on Scrapy. Since then Scrapy is continuously maintained and developed by this collaboration.

Spiders and items are the two main components of Scrapy. Spiders.py is a Python module where the information about item to be crawled/scraped, website to be crawled and other settings for spiders are mentioned. The scraped data are saved in the form of items. Items.py module contains information about items to be saved as fields. These items/fields are used by spiders to assign the scraped data.

3.1.1 Architecture of Scrapy

Figure 4 shows the major components of Scrapy and the flow of data represented by green arrow.

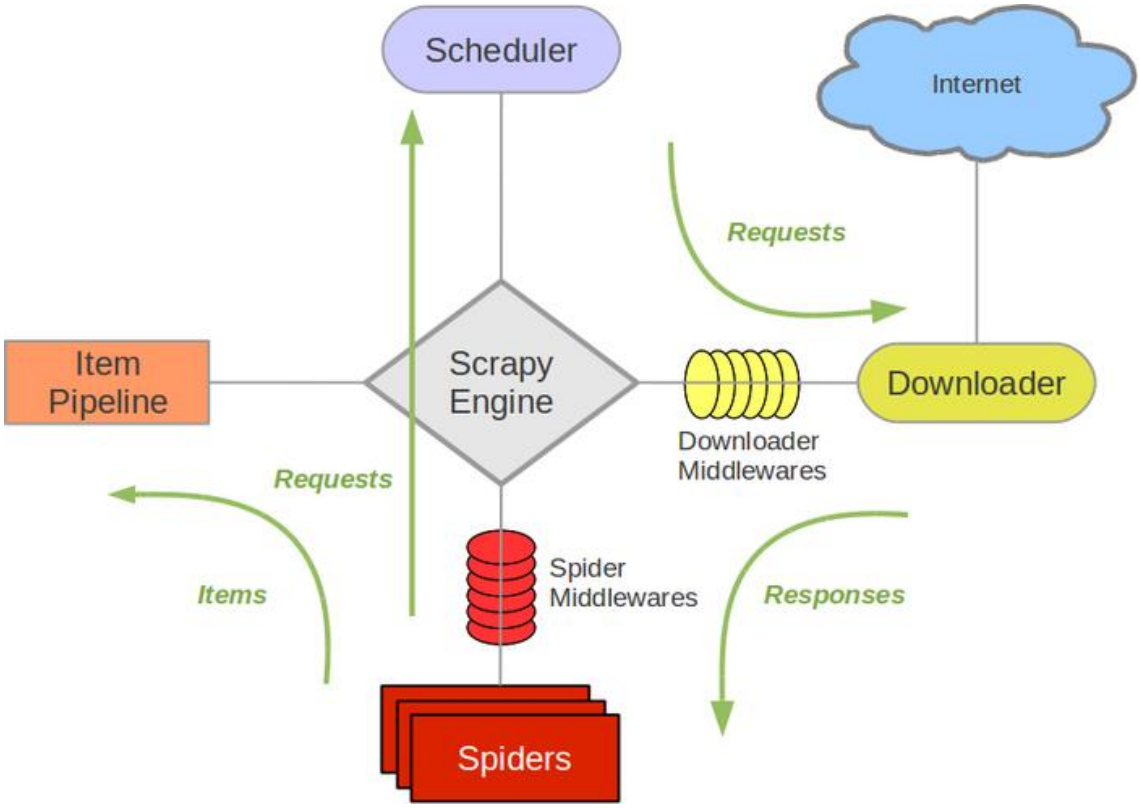

Figure 4. Components of Scrapy [7]

1. Engine

Scrapy Engine lies in the heart of Scrapy. It is the main component of this framework which controls the overall process in Scrapy and is responsible for how Scrapy functions. It controls the responses and requests that need to be passed from one component to another. It is responsible for triggering events when the other components perform certain action. All in all, it maintains the flow of data among the building blocks of Scrapy.

2. Downloader

It interacts with the internet and downloads the pages that are passed by Scrapy engine to it for downloading. The downloaded responses are passed back to Scrapy engine via downloader middleware.

3. Downloader Middleware

This component sits between engine and downloader and works as a bridge to pass requests from engine to downloader and responses from downloader to engine. It is used to modify how the responses and requests are passed between engine and downloader. Although it comes with a default behavior, it can be altered to suit one's need.

4. Spider Middleware

It is a specific hook that bridges the Engine and the spiders which are responsible for passing requests from spider to engine and responses from engine to spiders for item extraction. It controls how spiders behave and how to process response and requests.

5. Spiders

Spiders are the Python class that defines which page or domain to crawl and the way of extracting the required items. Spiders generate requests that pass to engine which in turn reaches the downloader. Once spiders receive the desired response, the items are crawled/scraped and stored.

6. Item Pipelines

This component acts more like a filter. A user can define what to store and what to ignore in this class. Validation of items, checking if the items are already scraped or not and cleansing the items are basic functions of this component.

7. Scheduler

It controls the timing of response and requests in Scrapy. When it receives requests from engine that should be passed to downloader, it queues them in order and so do when responses are passed to the scheduler from the downloader.

3.1.2 Workflow of Scrapy

The workflow of the Scrapy is the interactions between the components mentioned. The data flows as shown by the green arrow among these components.

Initially, the Engine opens the domain and finds the related spider for that domain. It will then find out the first URL to be scraped from the lists of URLs and sends it to the scheduler. Next the scheduler returns the next URLs to crawl requested by Engine which is passed to the downloader via the downloader middleware as request. The downloader then downloads the web page and once finished, it generates the response. This response now is passed in opposite direction of request, i.e., to the Engine via downloader middleware. The Engine feeds the spider with the response where it is processed to extract the required items. The Spider then will return the scraped items and new URLs to follow. The returned items are passed to the item pipeline where they are filtered and then stored in database while the URLs (new requests) are passed to the Engine. This process keeps on continuing unless there are no more requests/URLs to crawl.

3.1.3 Advantages of Scrapy

Mozenda, a powerful commercial web scraping tool, is selected as alternative to Scrapy. Mozenda can be a great tool especially for marketing and sales plan since it has features such as forecasting and analyzing competitor price. However, it lacks a good documentation compared to Scrapy. Scrapy, on the other hand, is a free open source tool. Scrapy is supported in major operating systems which is a big advantage over Mozenda supporting only Windows.

Scrapy is completely written in Python and is portable. Third party plugins can be hooked with it for extensibility without editing any of existing code. It takes less than few minutes to write a spider to scrape and save information and images. It also servers the purpose of data mining, monitoring and automation testing tool.

3.1.4 Limitations of Scrapy

AJAX components are challenges to Scrapy. It fails to download the dynamically loaded contents. Although AJAX contents can be downloaded using Selenium along with Scrapy, the web driver takes long to load the download contents. The installation process, which is fairly easy on other tools, and the project layout might be confusing for beginners in Scrapy .

3.2 Django

Django is an open source and free web framework developed in 2003 which is completely written in the Python programming language. It was created by Adrian Holovaty and Simon Willison at Lawrence Journal-World newspaper and named it after the famous guitarist Django Reindhart. Django Software Foundation was established in 2008 in order to maintain Django. The modules can be re-used again and again in Django allowing the rapid development of web programming and thus have a principle of “Don’t Repeat Yourself”. By now Django is well established framework with notable uses by Mozilla, Disqus, Instragram, Pinterest and lots of other famous websites.

3.2.1 Architecture of Django

Django uses a Model View Template (MVT) as software architecture in its core. MVT is software architectural pattern that divides the given software into three parts Model, View and Template. Figure 5 shows the components of MVT architecture along with the collaboration among its components.

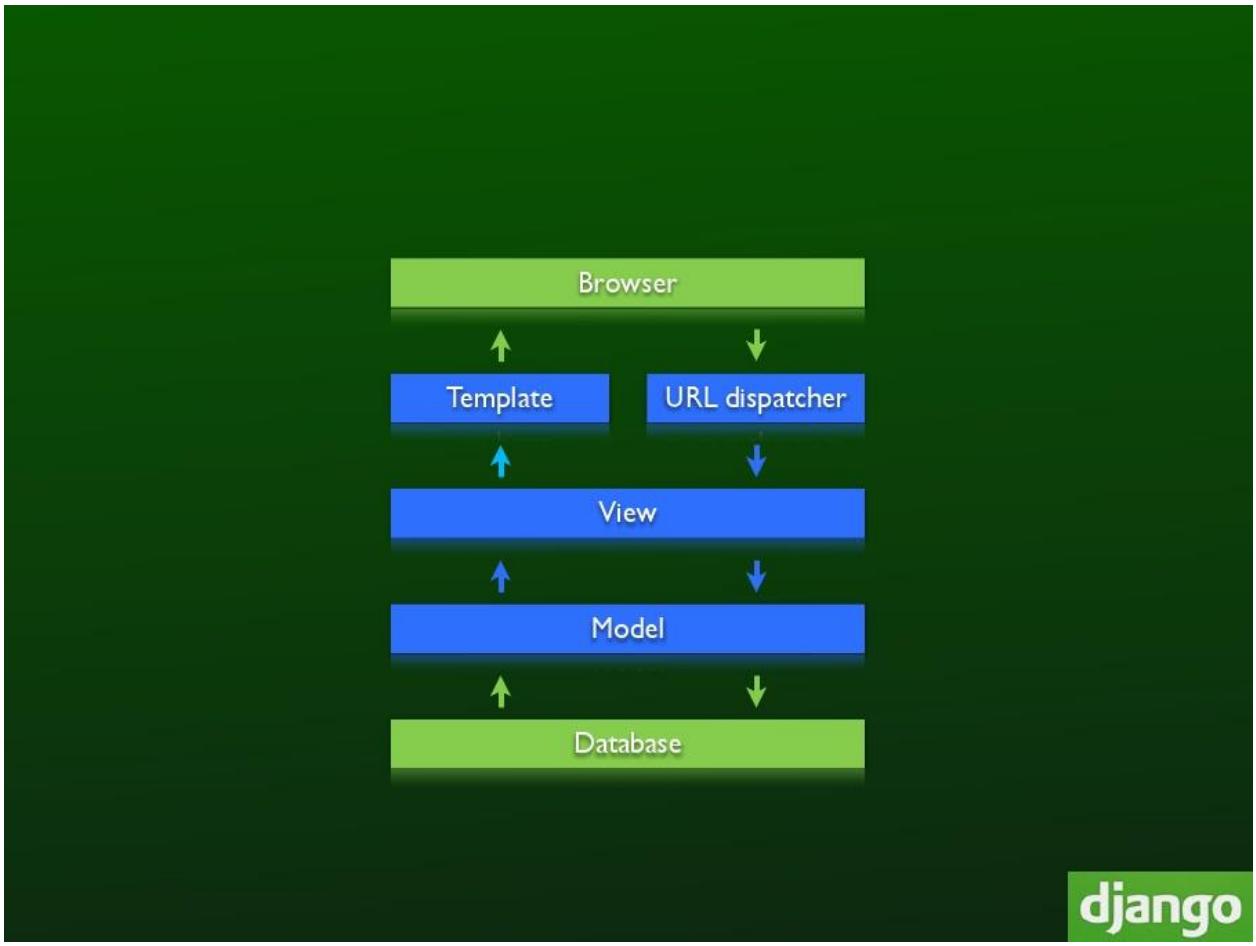

Figure 5. Architecture of MTV [8]

MVC (Model View Controller) is an extensively used architecture over multiple programming languages which makes codes mode maintainable. MVT is a modified version of MVC. Unlike MVC where model has the state, MVT have data. The components of MVT are discussed below.

1. Model

Model represents the database of application. It allows to create new database objects. In Django, the models.py module works as model. A model can be created as a definition of class with the attributes. Figure 6 shows a simple model declaration which creates a table called SampleModel with two columns title and body as char and text field.

```
class SampleModel(models.Model):
        title=models.CharField(max_length=120)
        body=models.TextField()
```

Figure 6. Defining Django models

2. View

View is the output that is presented to the users and how users interact with it. The view consists of HTML, CSS, JavaScript. In Django, views are defined as either class or function in views.py module. Class-based views can benefit from Django's in-built gridview (to display grid based view), listview (to render lists), or templateview (to render templates). Figure 7 shows a function based view which will render a template titles.html.

```
def title_view(request):
    titles=SampleModel.objects.all()
  return render_to_response("titles.html")
```

Figure 7. Defining views in Django

3. Template

Template describes how a user sees the view. These files are saved as .html files that consist of tags and filters. A template can inherit properties and attributes from another template making it Object-Oriented. It is a good practice to separate template files from other Django files in order to have a clean code and project. For this purpose a new template directory is created at the same level as that of application folder and .html files are created inside this folder. The newly created template folder should be mentioned in Django settings file so that Django knows about their location and how to render them. A simple template files looks like the code below which will list all the titles from the database.

```
{%extends "base.html"%}
<html>
  <head>
  <title>My titles</title>
  {%for title in titles%}
  <li>title.titles</li>
```

Figure 8. Defining Django template

URL Dispatcher and the Django Framework act as controller. It maps the URLs with the correct views so that the correct template is rendered. The URL Dispatcher is a urls.py module where the function url() is defined. A simple url() is composed of two arguments; the first argument is the regular expression that is matched from user input to render the correct view, while the second argument is the view associated with that particular pattern of url. The second argument is a call to the classes from views.py.

3.2.2 Workflow of Django

Once all the database, views and URLs are defined in Django, the workflow works as follows.

1. Initially it consults the URL patterns mentioned in the urls.py modules. This module contains regular expressions to capture the URL pattern to render the correct view.

2. Once the input is matched against the URL pattern and the correct URL is found, it calls the related view. The view will process the request and queries the database if needed.

3. Then the view passes the requested information to the correct template.

3.2.3 Advantages of Django

Django is a web framework so developers can create web sites with minimum effort very quickly as most needed hooks, such as templating languare and ORM are already attached. By now Django is well established and has the largest community of all frameworks therefore online help and suggestions with the projects come almost immediately. It has class-based views which makes it more Object-Oriented. Django has one of the best documentation available in the internet which is great for new users. The admin interface of Django allows the management of all the contents of web site from a single page for which no coding is required at all. All the databases of Django can be accessed without knowing SQL. Databases can be defined using python classes and accessed using Python with ORM of Django.

3.2.4 Limitations of Django

One of the flaws of Django is its templating system. Besides everything being pythonic, one cannot access the python function from template and this requires filter classes as solution. Django is evaluated as not performing well with client-side HTML5 apps as well as having poor ORM.

3.3 Dynamic Django Scraper (DDS)

DDS is a Django application based on Scrapy where all the crawling and scraping can be controlled via graphical user interface. For this project, DDS was used as it allow to control scrapers, scheduling and other tasks dynamically through Django admin interface. Although Scrapy might seem slightly daunting for the new users, the same tasks can be performed

with DDS with minimal coding. DDS was developed by Holger Drewes as Django project in 2011.

Since DDS is based on Scrapy, it supports most of the features supported in Scrapy such as item pipelines, regular expressions etc. However, this does not mean that Scrapy is not needed if DDS is installed on a machine. DDS works only when Scrapy is installed and all spiders and pipelines are defined and written in the Scrapy project. DDS makes it much easier to manage the Spiders via the admin interface. For example, the xpaths are hardcoded into programs in Scrapy while the same process can be achieved in DDS via the admin panel graphically.

3.3.1 Architecture of DDS

DDS consists of the following components.

1. Django Models

This module is similar to the models defined in Django for databases. In order to crawl and scrape data, three modal classes should be defined. One model class stores information about which site to crawl, the scraper runtime information, which scraper/spider to use and the second class that mentions what items are to be scraped. The third class is used from Scrapy that stores the extracted data and links them to the second class. Figure 9 demonstrates the Django model required for DDS.

```
from django.db import models
from dynamic_scraper.models import Scraper, SchedulerRuntime
from scrapy.contrib.djangoitem import DjangoItem

class NewsWebsite(models.Model):
    name = models.CharField(max_length=200)
    url = models.URLField()
    scraper = models.ForeignKey(Scraper, blank=True, null=True, on_delete=models.SET_NULL)
    scraper_runtime = models.ForeignKey(SchedulerRuntime, blank=True, null=True,
on_delete=models.SET_NULL)
    def __unicode__(self):
        return self.name


class Article(models.Model):
    title = models.CharField(max_length=200)
    news_website = models.ForeignKey(NewsWebsite)
    description = models.TextField(blank=True)
    url = models.URLField()
    checker_runtime = models.ForeignKey(SchedulerRuntime, blank=True, null=True,
on_delete=models.SET_NULL)

    def __unicode__(self):
        return self.title
class ArticleItem(DjangoItem):
    django_model = Article
```

Figure 9. Defining Django models in Dynamic Django Scraper [13]

In Figure 9, the first modal class stores information about scraper; its name, the site to crawl and runtime for scraper are saved. These fields can be filled via the admin interface later with an attractive GUI. The second class defines where to save the scraped items. Once the title is scraped from the URL, it is saved under the title column of this class. Checker_runtime tells which model should be checked to see if the duplicate items exist which is configured again in the admin interface. The final class is borrowed from Scrapy where the items are stored. It points to Django model to determine which fields are to be used to store the extracted items.

Next the scraped object classes are defined in the admin panel. Here fields / datasets that are defined in previous class are repeated as attributes. These attributes/datasets are available from scraper panel (also in admin interface) easily later to define their location in the document. These datasets have their type. All scraped object class need to have ‘base’

dataset that is marked as BASE for attribute type that defines the location of other elements in XML document relative to this base element. The base attribute works as an artificial construct and comes handy when the items to be scraped are inside one main block. Other attribute includes STANDARD (normal data), IMAGE (for images), DETAIL_PAGE_URL (pages that needs to be followed). Figure 10 shows a snapshot of scraped object class.

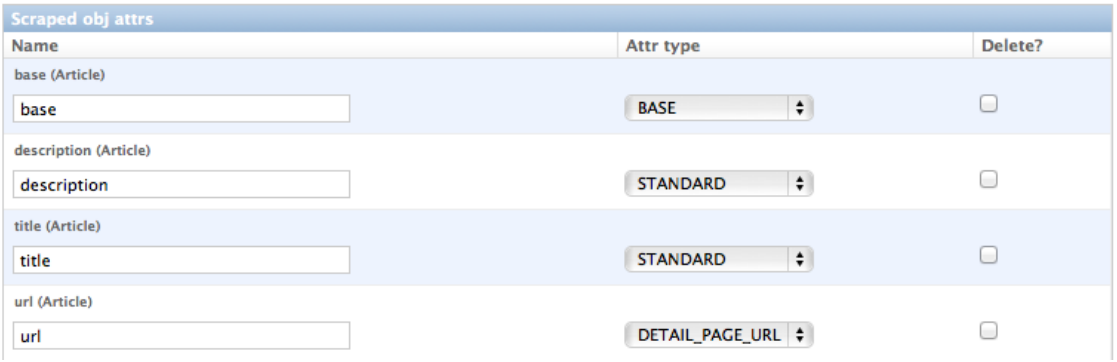

Figure 10. Snapshot of scraped object class [13]

2. Scrapers/Spiders

The spiders/scrapers are defined in spiders.py module. All the information that Scrapy needs from a spider are provided via classes. The predefined Django models are provided to the scrapers so the spiders know which site to crawl, where to store extracted items and which model to use. Figure 11 shows Spider linked with Django models defined in Figure 9.

```
class ArticleSpider(DjangoSpider):

    name = 'article_spider'

    def __init__(self, *args, **kwargs):
        self._set_ref_object(NewsWebsite, **kwargs)
        self.scraper = self.ref_object.scraper
        self.scrape_url = self.ref_object.url
        self.scheduler_runtime = self.ref_object.scraper_runtime
        self.scraped_obj_class = Article
        self.scraped_obj_item_class = ArticleItem
        super(ArticleSpider, self).__init__(self, *args, **kwargs)
```

Figure 11. Defining scrapers in DDS [13]

The xpath to the items are assigned from the admin panel.

3.3.2 Workflow of DDS

Once all the setup is ready for DDS, the scrapers are run either from command line or from celery as tasks. Since the scraper is provided with all the required information, it knows which domain to start to crawl. The mentioned web page is downloaded as DOM and parsed to spiders. Next the spiders extract the required items from the DOM and are passed through the item pipelines. Itempipelines cleanse the downloaded data and decide whether to save it or not based on rules and functions. Finally, the data is saved to database once they pass through the item pipeline.

3.3.3 Advantages of DDS

DDS does the same thing that Django and Scrapy do seperately. The main advantage of DDS is that the same module works for both Django and Scrapy and does not need more codes. Similary, the xpaths to the items can be managed from admin panel which is hardcoded in Scrapy. Instead of managing spiders from command line, they can be monitored from admin interface in DDS which can be done by users with less technical programming knowledge. DDS can be made extensible by plugging other plugins such as haystack (search database for Django).

3.3.4 Limitations of DDS

DDS supports filters that refine items before being saved to database. However, the filters defined by users seem to slow down the performance of DDS. Compared to other Python packages, DDS has a poor documentation.

3.4 Celery

Celery is a Python-based task scheduler framework. It is open source framework that is capable of asynchronous task scheduling (running the tasks in the background) as well as synchronously (waiting until ready).

The retailers from whom the information is scraped for this thesis is updated every few days and it is time consuming to wait until the update is made and run scrapers/spiders manually to scrape items. Queueing of tasks comes handy in situation like this. The Celery package is

integrated with Django and called Django-celery. In this project, Django-celery works as task scheduler which runs the spiders every week to scrape new contents from the retailers. In addition, it erases the previous scraped sale items from the database to make space for new scraped item.

Celery can be used for any task queueing purpose that runs asynchronously or in real time. It is also suitable for queueing heavy or long running jobs such as generating PDFs, resizing images, transcoding videos etc.

3.4.1 Architecture of Celery

Celery consists of four main components namely broker, worker, task and backend. Figure 12 shows these components and how they interact followed by a brief introduction of each component.

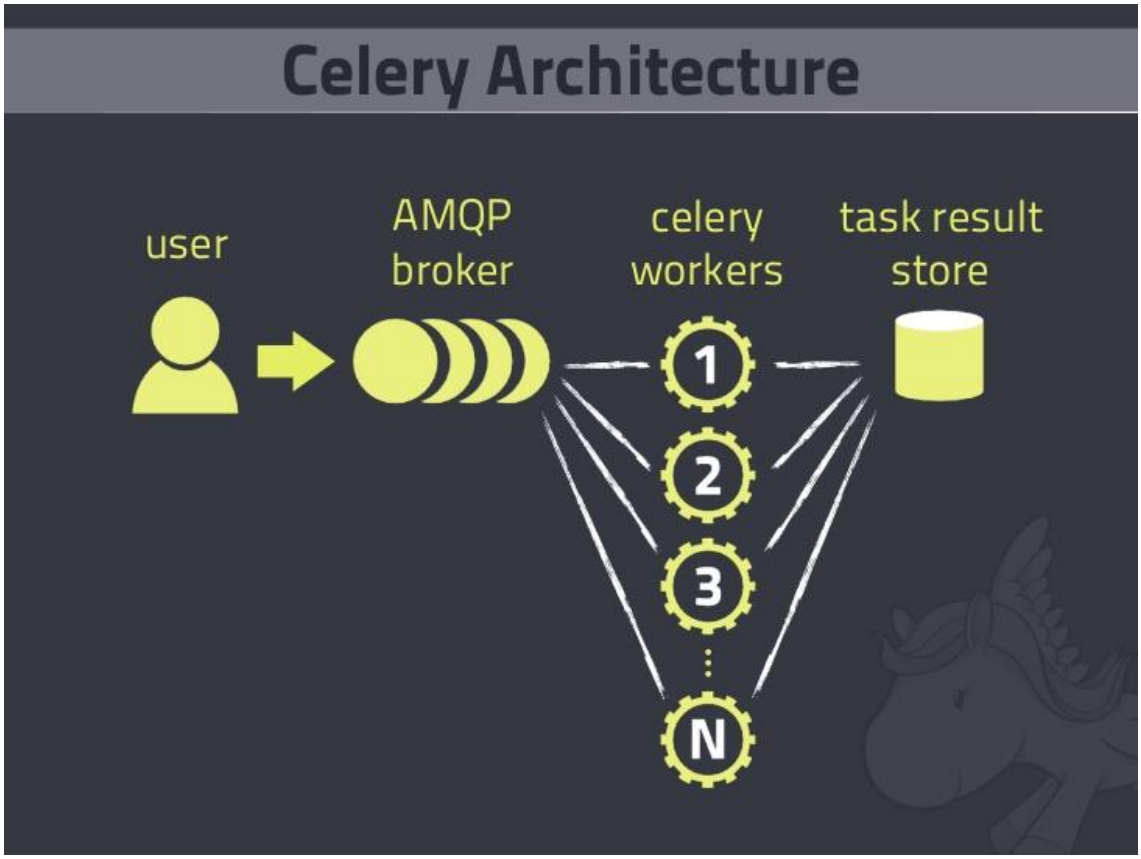

Figure 12. Components of Celery [11]

1. Brokers

Brokers are the software services that tell the workers what needs to be done. They work as a middleman between applications and workers. There are several brokers available to be used with Django for different purposes. Some of the popular brokers are listed below.

Combu is used as default message broker in Django-celery since its relatively easy to configure and it is automatically installed as dependency with Django-celery.

1.1 RabbitMQ

It is an open source messaging queueing protocol which is installed by default with Celery. It is compatible with major operating systems. It connects different servers and facilitates the communication between them either in local LAN or across Internet. It allows the queueing of messages to be mirrored across several machines for message redundancy, supports several programming languages, has user friendly management interface from where the brokers can be easily monitored. In addition, it is highly reliable and runs on several machines to form a cluster which behaves as a single broker.

Since RabbitMQ is the default message broker in Celery there is no need to install it.

1.2 Redis

It is another popular AMQP. It is easy to configure and use however is vulnerable to data loss in case of power failure.

2. Workers

Workers are the actual process that takes jobs from broker and perform the task that is required. More than one worker can be run in parallel to perform the jobs at high speed.

3. Tasks

Tasks are the building blocks of job scheduling/queueing. They are Python classes which are unique to each other. Each task is associated with worker which will get executed as per instruction. Tasks are separated from other modules to make the project/code more neat and usually written under tasks.py in Celery. Depending on the stage/time of execution, each task has its own state. The state of tasks change as it transits from one stage to another and the previous sate is replaced by the current state. The possible state of task can be one of the following.

Pending: the task is still in queue to be executed by workers.

Started: the task is carried out by workers but has not finished yet.

Retry: the task starts but fails to complete and is waiting to be retried.

Failure: the task starts but fails to complete due to exception or exceeding retry limit

Success: the task is completes as per requirement.

4. Backends

Backends are the storage system that is required to monitor the tasks and keep track of their states as well as store the values returned by the tasks. There are several backends to choose from depending upon the scale of project. Some of the common result backends used with Celery are RabbitMQ(amqp), MongoDB, Redis etc. Users can also define custom backends to suit their needs.

3.4.2 Workflow of Celery

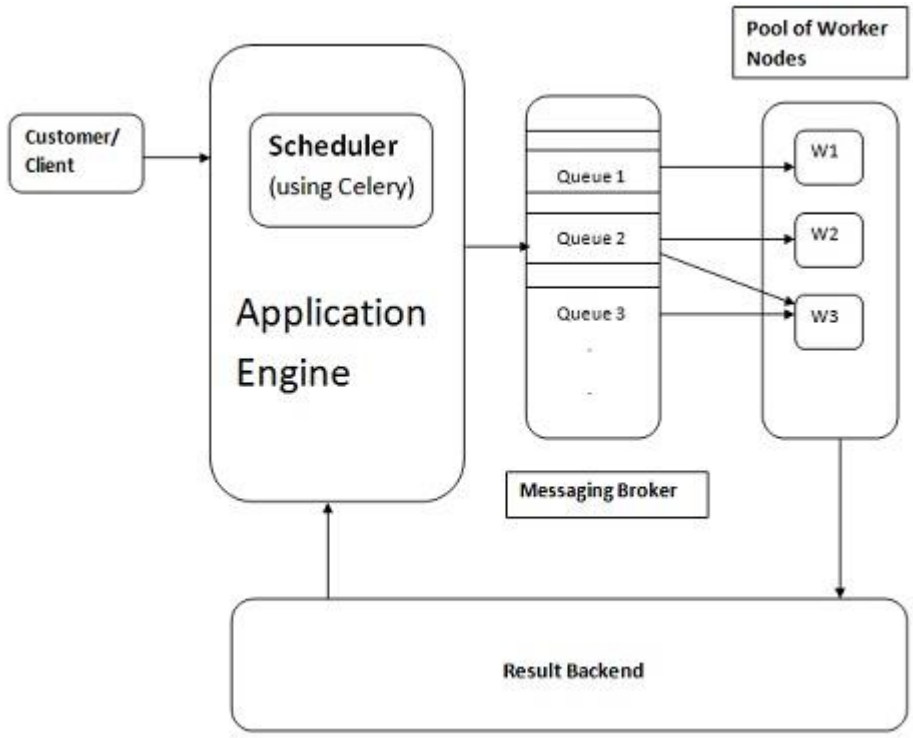

Figure 13. Workflow of celery [9]

Figure 13 shows the workflow of celery. When a user runs application, it contains tasks or task sets as a message. These tasks are pushed to brokers using AMQP. Brokers are the middlemen between users and workers. They in turn assign these tasks to each worker and keep track of them. The tasks will be shared among the workers as per the algorithm of scheduling to achieve faster performance. The workers execute the tasks and yield the result to a backend database if successful or else it will send the state of task so that it can be monitored. Finally, the user can easily interact with the database (Result backend) to

view the result. However, task scheduling is much easier with DDS. Using the DDS tasks can be monitored via the admin interface of Django. Figure 14 is a snapshot of how tasks can be scheduled in Django admin interface.

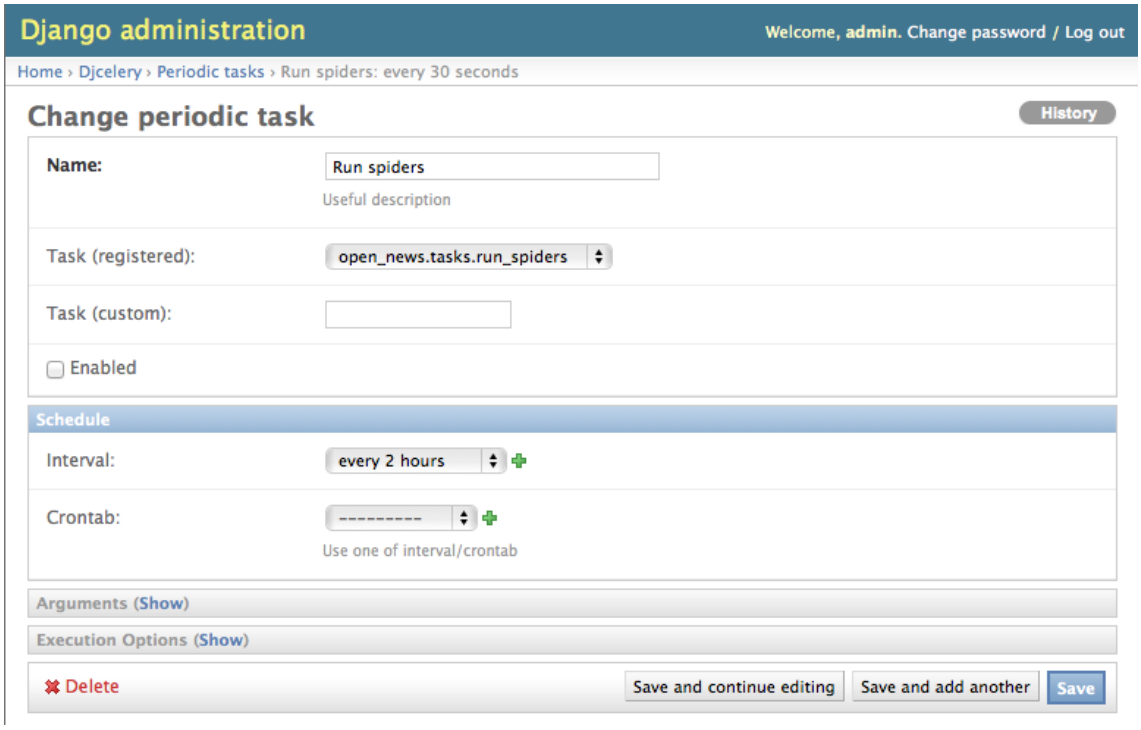

Figure 14. Controlling tasks via admin panel [10]

3.4.3 Advantages of Celery

Celery is capable of queueing synchronous as well as asynchronous tasks. The tasks can be run on a single worker or distributed among workers for parallel processing. It has powerful and flexible interface for managing and monitoring tasks.

3.4.4 Limitations of Celery

Most of the users in the internet have complained about the complexity Celery creates. It has a poor documentation and steep learning curves. Cron is more preferable for small scale projects for these reasons.

3.5 Other Packages

In addition to above mentioned packages three other packages are used to complete the project. These packages are used to assist the major packages that are discussed before.

1. Haystack

A search engine provides fast and convenient way to locate items. Haystack provides search fuctionality in Django. It is a reusable application meant for Django application but also works with third party application. Haystack provides highlighting, faceting, spelling suggestions or “more like this” search features. Haystack provides a developer to choose between different search engines such as Xapian, Whoosh, Solr etc.

Haystack provides an easy search feature for websites however it is not a good solution for non-model-based data. Haystack also has the inability to exclude search results from the database.

2. South

South is schema and data migration tool for Django projects. It is an easy solution to migrate the data forward or backward in database schema. It has the features of automatically writing migrations once the models have changed, allowing the migration of Django application itself.

3. Selenium

Selenium is a web automation tool widely used to automate human browsing experience. The main purpose of selenium is to automate web applications for testing purpose, however, it can be used for other tasks. Selenium can be used to scrape web pages with dynamic contents loaded via AJAX or JavaScript. Since the Selenium web driver is not primarily intended to be used for scraping, the scraping process becomes much slower and takes long time when used with selenium web driver. Using the Selenium web driver also generates larger traffic than simple HTTP requests and thus consumes more system resources.

The next chapter describes the implementation of these tools to create the sale-based web site.

4 IMPLEMENTATION

In order to build the web site for the purpose of this thesis Siwa, Vappa Valinta and K-market (grocery markets of Finland) were chosen to demonstrate the scraping and crawling process. The Django framework was used as web framework along with Scrapy as scraping and crawling framework. This chapter describes the process of creating the sale based website.

4.1 Defining Django models

A virtual environment was created for the project which acts as standalone environment without interfering with the system environment. Django and Scrapy were installed in the virtual environment using the pip command before DDS could be installed. To start the scraping process first Django models were defined to store the information of items to be crawled, the source page of item and spiders to be used.

```
# This is the referenced object class for K-extra market
class KextraObj(models.Model):
        name = models.CharField(max_length=15)
        scrape_url = models.URLField()
        scraper = models.ForeignKey(Scraper,blank=True, null=True, on_delete=models.SET_NULL)
        scraper_runtime = models.ForeignKey(SchedulerRuntime, blank=True, null=True,
on_delete=models.SET_NULL)

        def __unicode__(self):
                return self.name



# This model class stores the required information from K-extra page
class KextraMod(models.Model):
        market = models.CharField(max_length=15, default="kextra")
        brand=models.CharField(max_length=20,null=True,blank=True)
        product_primary = models.CharField(max_length=170)
        product_secondary = models.CharField(max_length=170,null=True,blank=True)
        discount = models.CharField(max_length=8, null=True,blank=True)
        priceEuros = models.CharField(max_length=4)
        priceCents = models.CharField(max_length=4)
        url = models.ForeignKey(KextraObj, null=True, blank=True)
        image = models.CharField(max_length=200)
        unit = models.CharField(max_length=25, blank=True, null=True)
        valid_until = models.CharField(max_length=50)
        normalprice = models.CharField(max_length=120 , null=True,blank=True)
#This is the djangoItem to store scraped items into models
class KextraItem(DjangoItem):
        django_model = KextraMod
```

Figure 15. Models for K-extra market

Before beginning to scrape the items three Django models were defined. In Figure 15 KextraObj model stores the information about the name, the URL to scrape, the spider to be used and the information about when to run the scraper. The second model, KextraMod in Figure 15, defines what items are to be extracted from the web site. This model is available from the admin interface of DDS to be filled with correct values. KextraMod has 12 fields that stores the information about name of the market, brand of item, primary name of item, secondary name of item (if there are more than one name), available discount, the price in euros and cents, image of the item, number of units, the validity of discount and the normal price. The Xpath to these variables were configured from the admin panel of Django. Figure 16 shows the Xpath for the mentioned datasets. Finally, a third model was defined that actually connects the Django model with the Scrapy. In Figure 15 the KextraItem model informs Scrapy about which model to use for storing scraped information.

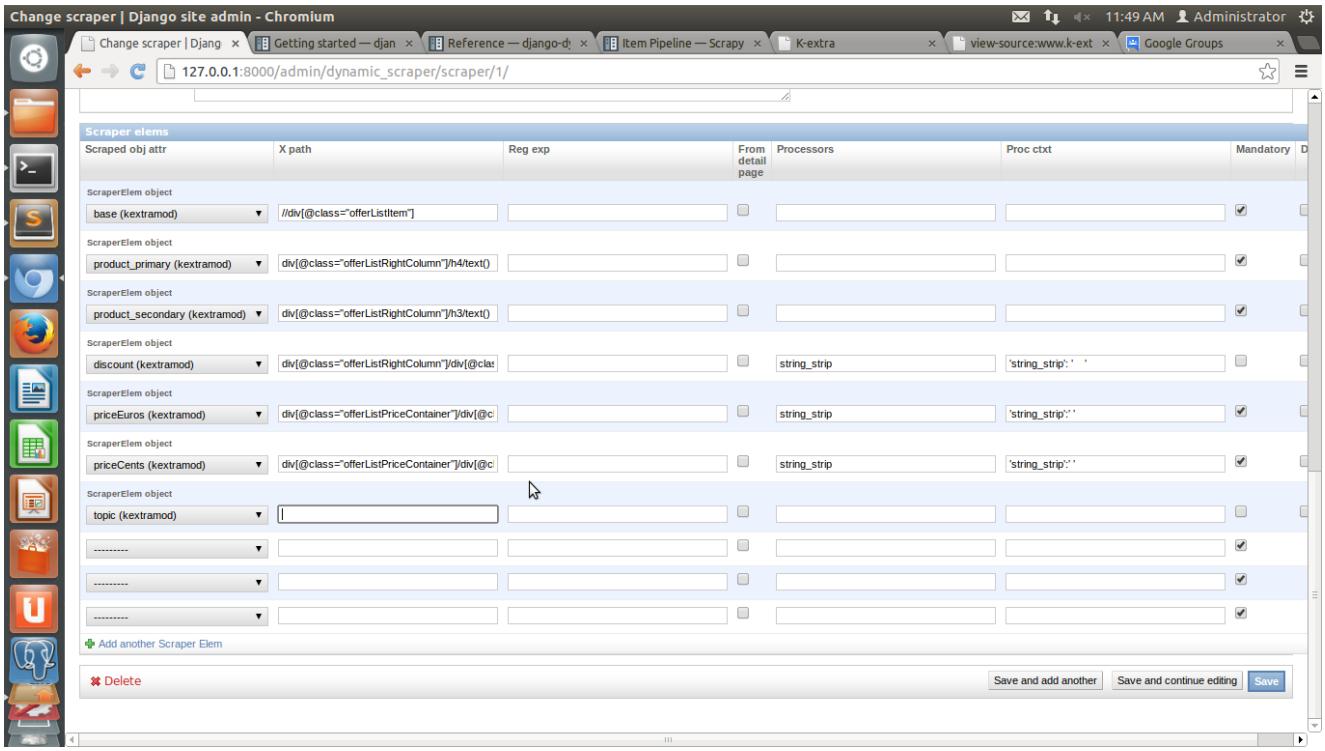

Figure 16. Defining Xpath to the datasets for KextraMod model

4.2 Defining spiders/scrapers

Next the Spiders were defined in Scrapy from where the web pages are requested. Figure 17 shows the KextraSpider that has information about the previously defined KextraMod and KextraItem models to scrape the items.

```
class KextraSpider(DjangoSpider):

    name = 'kextra_spider'

    def __init__(self, *args, **kwargs):
        self._set_ref_object(KextraObj, **kwargs)
        self.scrape_url = self.ref_object.scrape_url
        self.scraper = self.ref_object.scraper
        self.scheduler_runtime = self.ref_object.scraper_runtime
        self.scraped_obj_class = KextraMod
        self.scraped_obj_item_class = KextraItem
        super(KextraSpider, self).__init__(self, *args, **kwargs)
```

Figure 17. Defining spider class for K-extra market

4.3 Creating tasks in Celery

Then the tasks were created in Celery to automatically run it in the background. Figure 18 shows the KextraMod (the model to store the scraped data) is deleted first if there is any outdated data. Next the kextra_spider (spider) created in Figure 17 is run to scrape items. This task was available through Django admin panel to configure the time interval to run again.

```
@task()
def kextra_task():
        t=TaskUtils()
        KextraMod.objects.all().delete()
        t.run_spiders(KextraObj, 'scraper', 'scraper_runtime', 'kextra_spider')
```

Figure 18. Creating tasks in celery for K-extra market

Finally the task was executed from the admin panel of DDS and the items were scraped and stored in local host.

4.4 Scraping dynamic content using Selenium web driver

The locations of markets used for the project were scraped from http://www.fonecta.fi/ whose contents were loaded via AJAX. The Selenium web driver was used to scrape this dynamic web content since it was not available through normal scraping. In Figure 19 first, Selenium web driver and the Spider were initialized. Next, all the links to the individual markets were scraped and stored in 'links' variable. Then the individual web sites for each market were browsed by Selenium web driver to scrape the address of each market and were stored in 'address' field. Finally, the web driver closed when all the links were scraped.

```
class locationkextra(InitSpider):
  name="kextraloc"
  start_urls=["http://www.fonecta.fi/yritykset/haku/-/K-extra/-/5"]

  def __init__(self):
     InitSpider.__init__(self)
     self.browser=webdriver.Firefox()

  def parse(self,response):
     self.browser.get(response.url)
     time.sleep(5)
     sel = Selector(text=self.browser.page_source)
     links = sel.xpath('//ol[@class="listing"]/li/h4/a/@href').extract()

     for link in links:
        prefix="http://www.fonecta.fi/"
        link = prefix+link
        self.browser.get(link)
        sel=Selector(text=self.browser.page_source)
        l = ItemLoader(locationItem(),sel)
        l.default_output_processor = Join()
        l.add_xpath('address','//strong[@itemprop="streetAddress"]/text()')
        l.add_xpath('address','//span[@itemprop="postalCode"]/text()')
        l.add_xpath('address','//span[@itemprop="addressLocality"]/text()')
        time.sleep(9)
        yield l.load_item()
     self.browser.close()
```

Figure 19. Scraping dynamic web content using Selenium web driver

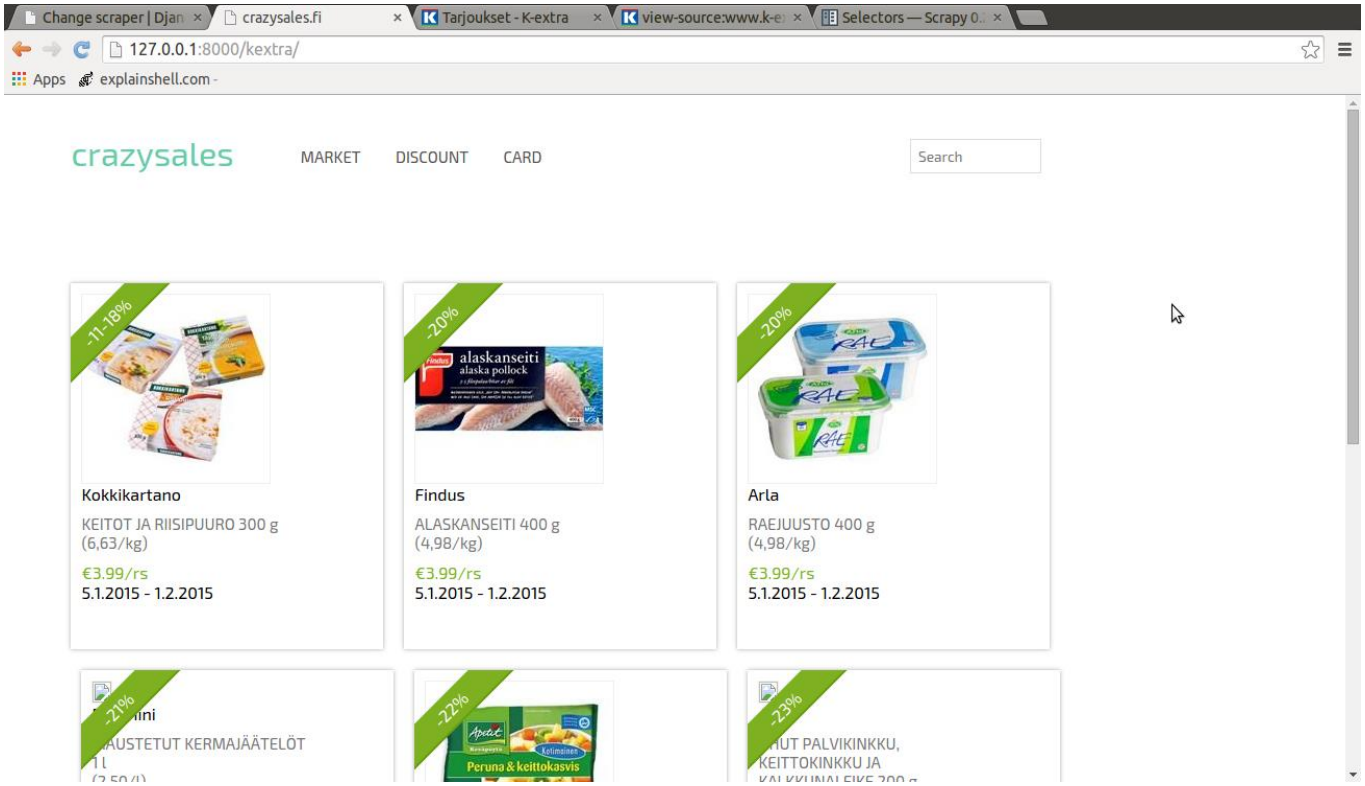

Figure 20. Snapshot of sale based web site for K-extra market

Figure 20 shows the snapshot of sale-based web site that displays the image of the product , its basic information along with price and validity of discount.

5 RESULTS

Web scraping is a tedious process if the website being crawled does not provide any RSS feeds. All the web pages that are crawled to build the sale-based website do not have any RSS feeds and their structure change frequently. Therefore, the changes have to be made often in database and Xpath. The job of scraping would be much easier if there were RSS feeds. Next, Scrapy has difficulties downloading the images. The downloaded images are saved with the filename of md5 hash. The images cannot be rendered from Django because Django refers to images with the original filename instead of md5 hash. The problem can be solved by using a pipeline but it makes the process slower.

Similary, the Selenium web driver fails to download the contents of the web page with the 'range' function. For example, if there is a web site with a page number at the end that starts with 1 and ends at 3 such as www.turkuamk.fi/index/1 and the last page is www.turkuamk.fi/index/3 . Using the range function to download the contents makes the Spider to drop items to be scraped, which is not desirable.

The web site built can be improved with an additional feature of sale alert. This feature allows consumers to save certain items on their preference list. As soon as the item is on discount, consumers can be notified via email or other form of message. In addition, instead of just displaying the locations of nearby markets, images of highly discounted items available on that market can be displayed in Google maps.

6 CONCLUSION

The process of following links to create the web index is called web crawling. Web robots, crawlers are used for crawling. Similarly, the process of simulating the human exploration of websites to extract information is called web scraping. Web scrapers are used for scraping. The objective of this thesis was to develop a sale-based web site whose contents is updated via scraping and crawling. Scrapy is used as a web crawling and scraping tool with a Django framework to develop the web page. Celery was used to schedule and repeat the crawling and scraping jobs.

Web sites of popular grocery stores of Finland, such as K-market, K-super market, Siwa etc. are the source of sale information which are displayed in grid format. The sale-based web site also allows users to view the markets that are near their location using Google maps API.

This thesis provided an introduction to web crawling and web scraping and their purpose. It also introduced tools such as django, scrapy, celery and dynamic Django scraper and techniques for scraping and crawling. The developed sale-based web site successfully demonstrates that web scrapers and crawlers can be used to extract various kinds of information from the web.

REFERENCES

[1] Rebecca Smithers, “Online discount vouchers increasing in popularity, consumer websites report”, [online]. Available: http://www.theguardian.com/money/2010/jun/16/money-off-discount-coupons-vouchers [Accessed 15 September 2014]

[2] “Web Crawler”, [online]. Available: Wikipedia, http://en.wikipedia.org/wiki/Web_crawler [Accessed 15 September 2014]

[3] “bandwidth reporting –cPanel”, [online]. Available: http://forums.cpanel.net/f5/bandwidth-reporting-323082.html [Accessed 28 October 2014]

[4] “Total number of Websites-Internet Live Stats”, [online]. Available: http://www.internetlivestats.com/total-number-of-websites/ [Accessed 28 October 2014]

[5] “Web Crawler”, [online]. Avaiable: http://en.wikipedia.org/wiki/Web_crawler#cite_note-cho2003-28 [Accessed 15 September 2014]

[6] “Web scraping”, [online]. Avaiable: http://en.wikipedia.org/wiki/Web_scraping [Accessed 20 September 2014]

[7] “Architecture overview-Scrapy 0.24.4 documentation”, [online]. Avaiable: http://doc.scrapy.org/en/latest/topics/architecture.html [Accessed 3 October 2014]

[8] Adzmely Mansor, “Django/Python Framework”, [online]. Avaiable: http://www.scribd.com/doc/49180165/Django-Python-Framework#force_seo [Accessed 6 October 2014]

[9] ”C-SyTS Cloud based multi-tasking System for Task Scheduling”, [online]. Avaiable: http://www.cs.ucsb.edu/~sivabalan/cs263/ [Accessed 15 October 2014]

[10] “Advanced topics--django-dynamic scraper 0.3 – alpha documentation”, [online]. Avaiable: https://django-dynamic-scraper.readthedocs.org/en/latest/advanced_topics.html [Accessed 23 October 2014]

[11] Idan Gazit, “An Introduction to Celery”, [online]. Avaiable: http://www.slideshare.net/idangazit/an-introduction-to-celery [Accessed 3 November 2014]

[12] “Breadth-first search”, [online]. Avaiable: http://en.wikipedia.org/wiki/Breadth-first_search [Accessed 15 September 2014]

[13] “Getting started--django-dynamic scraper 0.3 – alpha documentation”, [online]. Avaiable: https://django-dynamic-scraper.readthedocs.org/en/latest/getting_started.html [Accessed 23 October 2014]

[14] “Django documentation | Django documentation | Django”, [online]. Avaiable: https://docs.djangoproject.com/en/1.6/ [Accessed 5 November 2014]

[15] “Stackoverflow”, [online]. Avaiable: http://stackoverflow.com/ [Accessed 17 November 2014]

[17] “Celery – Distributed Task Queue – Celery 3.1.17 documentation”, [online]. Avaiable: https://celery.readthedocs.org/en/latest/ [Accessed 8 November 2014]

[18] Google, “Google Developers”, [online]. Avaiable: https://developers.google.com/maps/ [Accessed 9 November 2014]

[19] “Selenium WebDriver – Selenium Documentation”, [online]. Avaiable: http://docs.seleniumhq.org/docs/03_webdriver.jsp [Accessed 17 November 2014]

[20] “XPath Tutorial”, [online]. Avaiable: http://www.w3schools.com/xpath/ [Accessed 24 October 2014]

[21] “Scrapy 0.16 documentation – Scrapy 0.16.5 documentation”, [online]. Avaiable: http://doc.scrapy.org/en/0.16/ [Accessed 6 November 2014]