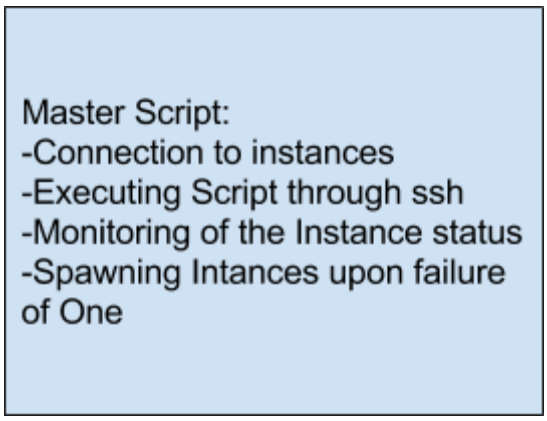
My Distributed Crawler Design
Sunny Singh
The Master Script Design



1. **Connection to the EC2 Instances:**

- Using boto library Ec2connection
- Using .ssh command to spawn my crawler.py py already at the AMI or by evertime pushing the script using user data field by passing inline parameters.

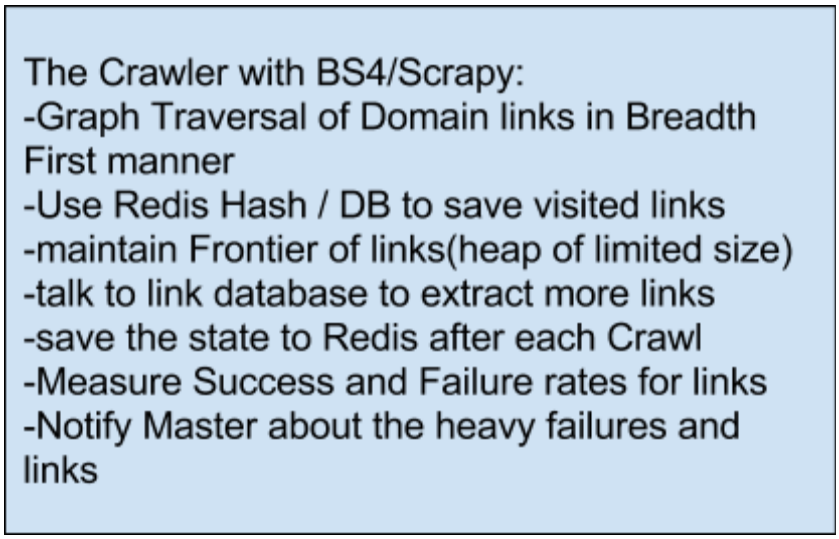
```
from boto.manage.cmdshell import sshclient_from_instancefrom
ssh_client = sshclient_from_instance(instance,
                    'Ec_2_key_pair.pem',
                    user_name='ec2-user')
status, stdout, stderr = ssh_client.run('sudo python crawler.py url_parameter item_parameter2')
```

- We will continuously loop through our instances and run following code with some delays using time library:

```
instance[i].update()
If instance[i].state == 'terminated' :
        spawninstance(instance[i] ,parameters)
#spawninstance will  restart the instance using ssh commands
```

- Use **CRON** to schedule our Run our server everyday

The Crawler Design



Idea is to crawl a website as **graph of web pages** with links as the edges of a graph using :

1.Breadth first traversal of a graph
2. Using hashes in redis / Bloom filter of pages to store the visited links
3. Queuing into a priority queue failed links along with current adjacent edges / or links temporarily unavailable till certain no of attempts.

My crawler will be specific to the structure of the website which i am crawling , Crawling will be done page by page, as going for item by item will be costly as it will need several In out data from the EC2 server , hence increasing the cost ,these are some important considerations

Maintaining url frontier and writing to Redis On regular intervals

1. we will add to url **frontier** only the internal links only to the website discovered at every page and at the same time remove the link that has been explored, to keep the no of links limited .

2.**Frontier** will be a **Priority Queue(using max heaps)** of certain length where each fresh link is assigned highest priority , a failed link is given certain negative priority and negative pripority accumulates , everytime there is link faliure.

Class item():
    Score ;
    link;

3. Links with some threshold no of failures will be given dead status and pushed out permanently to a bucket (database) .

```
This_link = frontier.pop()
Response = crawl_link(this_link.url)
if(response== 'failed'):
        this_link.scoe-= red_large
       if(this_link.score() < threshold):
            discard()
Else :
        this_link.scoe-= red_large
       push_to_db()
```

NOTE: We will maintain a timer and at each interval we will write frontier to the Redis as a snapshot : the hash of used links along with last writtenurl forntier.

4. Maintaining the size of frontier :

Extract older links from the database (stored somewhere at s3 server or RDS) ,(sorted by last used dates)
Suppose a MongoDB instance will store links and last used dates

```
Schema = {  _id: type_string(),
            date = datetime(),
             link_url=string(),
             website=string(),
          }
```

Whenever after popping from the frontier if the size of the frontier reduces less then 50 (assuming maximum size is 100) , we query 50 most transient links :

```
links = db.links.find().sort({'date':pymongo.DESCENDING})[0:50]
filter.push(links)
```

Maintaining the list of Discovered Links

1.Before moving to the new link it is necessary to check if the had been explored , this can be done using the **bloom filter (pybloomfiltermap)** library for python or **storing hashes** of the pages that have been crawled onto the redis server or into **mongoDB** .

2. Another way would be to Store the visited URLs in a single (a several bucketed) Sets for history lookup and auto deduplication. Use a Sorted Set with a URL's score set to the epoch value of its crawl time to have them ordered and do range queries.

Other Important Concerns and solutions:

Threading and Common Link collision

Going for threading for scrapping different domain items will be good thing to do in case if the websites being scrapped are huge .
In that case it is necessary to maintain the **Isolation among the domains** by limiting the next selected links from the current page and restricting them so that the page hit don't collide too often not doing so will increase redundancy and time required to scrape same data.

Mapping threads to Domains

In order to maintain that the threads are evenly assigned to various domain , we must choose the no of threads for a crawler to be coprime to the number of domains.

Observing Robots

Before moving on to each link for scrapping we will use RobotParser library to see if the robot exclusion protocolos valid. If for certain no of links robot disallows scrapping we can kill the scrapper by passing a response to master.

Master can maintain a list of heavily blocked urls that fail frequently and choose prefered url's before reinvoking the EC2 instance.

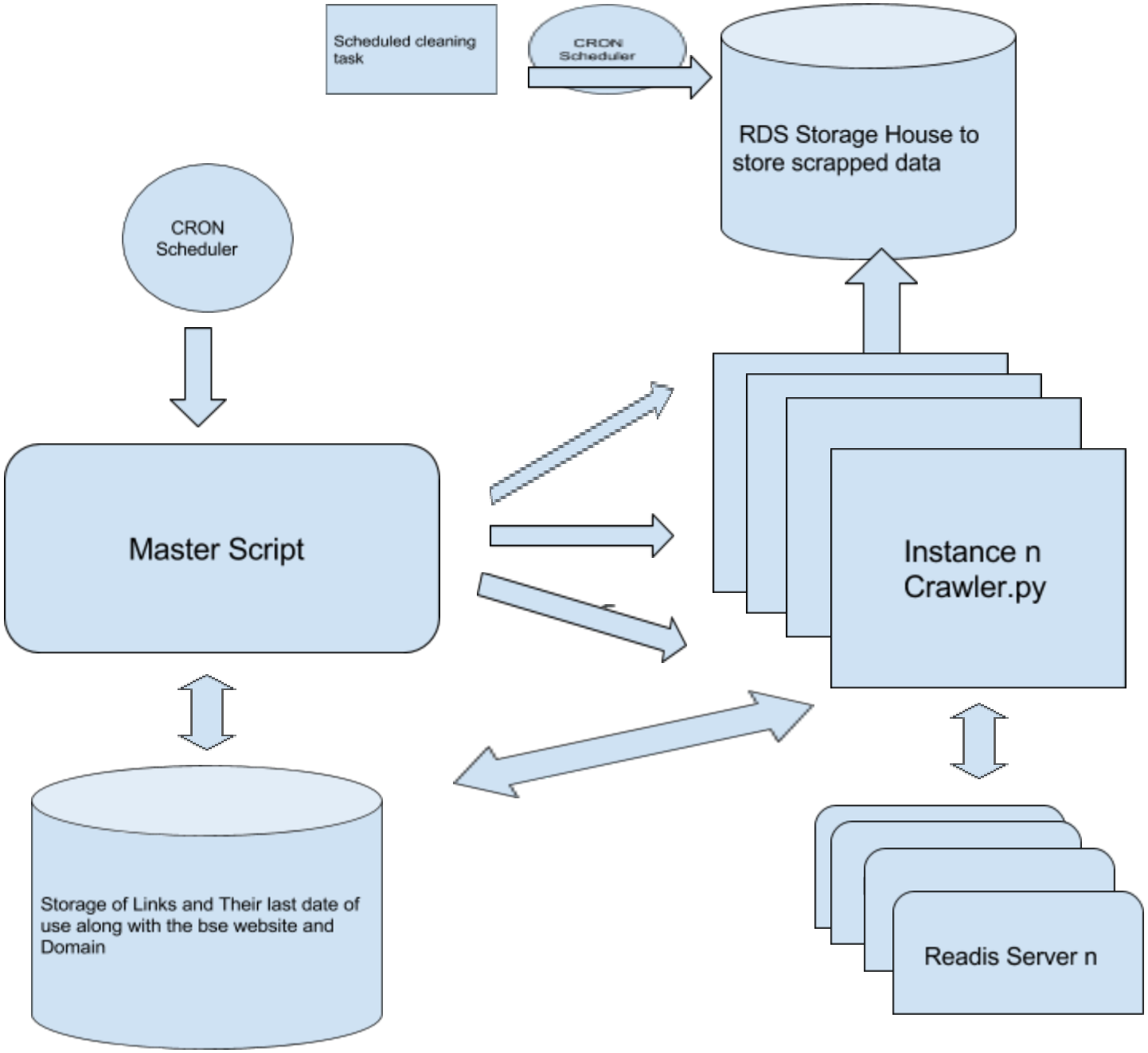
Items Storage :

The crawled data will be in the form of the Dictionary which can be directly written into MongoDB sharded database or a shared Relational Database such as Postgresql. Different shards can be assigned domains for write .

Amazon CloudWatch can be setup to notify you when usage patterns change or when you approach the capacity of your deployment, so that you can maintain system performance and availability.

schedule timely Discard Operation on the RDS database to remove the transient data that are no more useful of very very less frequently accessed as compared to others.

CRON can again be used to schedule this operation



Maintains the
frontier state and Hashes of the
visited links useful during failure.