Crawler Queuing and Priority Algorithms

# Introduction

The crawler queuing algorithm is the method used for determining which feed should be crawled next.

The priority algorithms have to do with determining when a feed thinks it should be crawled next.

# Overview

Crawling can either be performed:

* Continuously (a job is always being crawled, there is no down time).
  + Advantages:
    - Makes use of processing power (no wastage)
    - Less likely to end up with bottlenecks, since it will spread the load more
    - …
  + Disadvantages:
    - Hard to configure
    - Constant load on system
    - …
* Using essentially scheduling (a job is crawled when it hits its scheduled time, or at the next opportunity).
  + Advantages:
    - Less processing power used (only crawls when it really needs to)
    - Easy to rank crawls by priority and set schedules accordingly
    - Crawling schedules better suit the individual feeds
    - …
  + Disadvantages:
    - VERY hard to configure schedules
    - Can easily have overlapping schedules causing bottlenecks (also hard to predict when these will happen)
    - Crawlers will often (perhaps) be idle, which is a waste
    - …

Continuous probably the better choice of the two, since it will be easy to write a working queuing algorithm for.

The queuing algorithm:

* Will work around a priority queue system
  + Queue is a heap ranked firstly by ‘Priority’ (this could be a particular constant or a calculated value… see the below sections for more details) then by the time the site thinks it will be next updated
  + The ‘time the site thinks it will next be updated’ is a calculated value. It is calculated by using the time last crawled plus the average time taken for an update (i.e. the time between the last and previous update).
  + The average time taken for an update will be calculated by:
    - Keeping a variable for sum of total times crawled for that feed
    - Keeping a variable for the current average time between updates (minutes)
    - Then calculated by multiplying the sum by the current average, adding the last update time and dividing by the sum + 1
    - Gives a constantly updating average
    - Will be limited by the fact you can only store for a certain number of crawls…

New design decision:

Since crawler is continuous should use a continuous queue (i.e. a queue where the moment a job is finished it gets added straight back to the end of the queue). Queue is a number of jobs (aka crawls which need to be performed)

Queue is again a priority heap, priority calculated mainly from the equation:

Number of jobs currently waited for (since last update) DIVIDED BY Average number of jobs waited for between updates

This means the longer they wait the higher the priority, and the ones updated more frequently will more quickly raise in priority

When an item is added back into the queue it has a priority of 0, therefore it will never be crawled before the others

Has to be a special case where avg number of jobs waited for is 0, since this is infinite. Should only occur on first-run OR if all jobs can be executed at once (hence there is no need to wait)

(Priority will also factor in a number of other variables)

Calculations of priorities will have to do with:

* Calculation per iteration – calculates new avg if needed and updates priority
* Calculate for first run – will not use the general equation since there have been 0 jobs waited for, with all items. Will mainly go off just the avg and the type