Efficient Scheduling Algorithms for On-demand Wireless Data Broadcast

Στολτίδης Αλέξανδρος, Λάντζος Στέργιος

February 19, 2022

Contents

1 Introduction

With the rapid growth of mobile data services, there is an increasing need for scalable wireless data dissemination techniques which can concurrently deliver data to a large number of users. Wireless data broadcast is such a method to enhance the system scalability that allows multiple users to access data simultaneously. An important performance concern in wireless data broadcast systems is access latency, which is defined as the time duration from the moment that a request is submitted to the moment that all the requested data items are downloaded. In this project a new scheduling scheme for on-demand Wireless data broadcast is implemented. This scheme is suggested by the Efficient Scheduling Algorithms for On-demand Wireless Data Broadcast paper and includes three different algorithms that allows us to take advantage of the system's available throughput while also taking into consideration the frequency that each data item is requested.

2 System Model and Problem Statement

In order to implement the efficient scheduling algorithms mentioned in the given paper we must first recreate a virtual simulation environment to experiment and benchmark the given schedulers.

To begin with, a list of random data items are spawned. Each item has an id, a random size and an access probability. This list of items is used by the clients to select data items based on the Zipf's distribution. The range of size for the data list items is explicitly defined in the configuration file of the simulation routine. Each client selects a arbitrary number of requests to submit to the server. The minimum and maximum number of requests that each client can submit is also explicitly defined in the configuration file

The server takes each submitted request and creates a list of submitted request. The list of submitted requests is passed through a scheduler and a selected number of responses are served back to the clients. The scheduling process is consisted of two parts. The first part is the execution of the MTRS and a Pruning Algorithm to optimize for throughput on the downstream pipe. The second part takes into account the sequence that the requests must be served in order to optimize for latency on the client's side. The access latency is minimized using the MLRO Algorithm.

An image of the execution process and the flow of data from and to the server and the clients can be seen bellow.

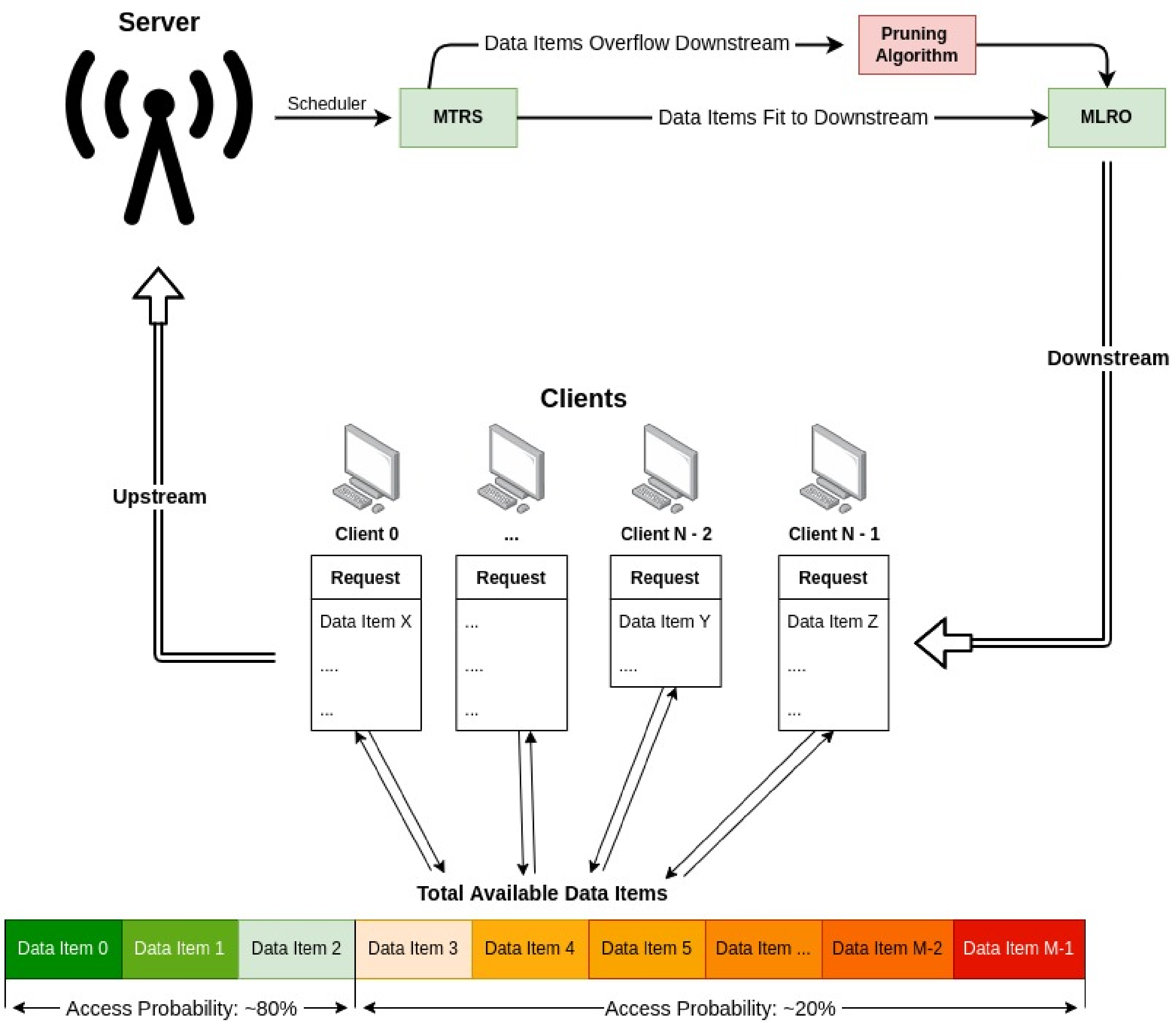


Figure 1: Data Items, Clients and Server

3 Maximum Throughput Request Selection (MTRS)

The MTRS scheduling scheme aims at finding a subset of pending requests to be broadcast in the next time period such that the throughput is maximized. In order to solve the MTRS in polynomial time we must find and select the data items d_j contained in a request Q_i such that the quotient below is maximized.

$$\frac{\sum_{Q_i} x_i}{\sum_{d_j} t(d_j) * y_j} \tag{1}$$

The x_i, y_j variables denote whether a request or a data item are selected. To further elaborate, when $x_i = 0$ the request is not selected and its data items can not be selected either. However, when $x_i = 1$ the request is selected and its data items can either be selected, $y_j = 1$, or not, $y_j = 0$.

We find the corresponding values of x_i and y_j that the above quotient is maximized with the help of linear programming by using Python's PuLP library. A pseudo-code of the MTRS algorithm is presented bellow.

```
Input: Received Requests (P) with Data Items (D)
Output: Optimal Requests (Q)

---------------------- Algorithm ----------------------
Q <- []
mark available requests with x <- 0
mark data items with y <- 0


find optimal solution with Equation (1)
mark requests with x <- 1 when attributing to optimal solution
mark data items with y <- 1 when included to an optimal request

append requests with data items to Q
return Q
```

4 Pruning Algorithm (Least Loss Heuristic)

Sometimes the set of requests that were chosen by the MTRS can not be sent in δ time slots and thus a few requests must be discarded at the time. These requests will either be sent later combined in different sets of requests, where the available time slots are sufficient, or they will be dropped completely by the server. In our implementation we set a large enough δ in order to serve all connected clients and not to discard any requests. A pseudo-code of Least Loss Heuristic can be seen below.

```
Input: Optimal Requests (Q) with Data Items (D)
Output: Optimal Requests (Q)

---------------------- Algorithm ----------------------
W <- Remainder data items
T(Q) <- Time to send Q


while (T(Q) > δ)
  d <- item that min(total requests containing d / time to send d)
  delete d from W
  delete request from Q if Q contains d
```

5 Minimum Latency Request Ordering (MLRO)

After all optimal requests with their data items are selected an optimal scheduling scheme must also be implemented in order to further optimize for access latency. With the help of dynamic programming an optimal schedule can be found and either the sequence of requests or their data items can be rearranged in order to prioritize for lowest latency. The MLRO pseudo-code is described below.

```
Input: Optimal Requests (Q) with Data Items (D)
Output: Optimal Schedule (S)

---------------------- Algorithm ----------------------
W <- Data Items of Q
n <- Length of Data Items


if $n \leq |Q|$
  find optimal schedule S for the data items
else
  find optimal schedule S for the requests

return S
```

6 Simulation Implementation

To further understand how these scheduling algorithms were implemented we briefly wanted to describe how the whole simulation was implemented. To begin with, each client runs in its own POSIX thread. In addition, the server, just like the clients, also runs in its own thread and is responsible for all the synchronization between threads. When a client submits a request the client thread blocks till notified by the server. After the server has processed the submitted requests it notifies all clients to check the responses located on the down stream. If a client was completely served the access latency is calculated and the thread finishes its execution. If a client was not served he blocks and waits till the next server notification. Finally, semaphores where used as a synchronization mechanism and the PuLP library for linear programming solutions in MTRS.

7 System Configuration

The system configuration that we used plays an important role in understanding how the scheduler performs in different scenarios and how the code was written to take advantage of the operating system utilities and the available hardware. Bellow a list of our configuration specifications is listed.

1. `Operating System`: Ubuntu 20.04.1
2. `Kernel Version`: 5.11.0-38-generic
3. `Python Version`: Python 3.8.10
4. `Processor`: AMD Ryzen 3900x (12 Physical Cores, 24 Logical Cores (Threads))
5. `Memory`: 16 GiB RAM at 3200 MHz

8 Statistical Analysis

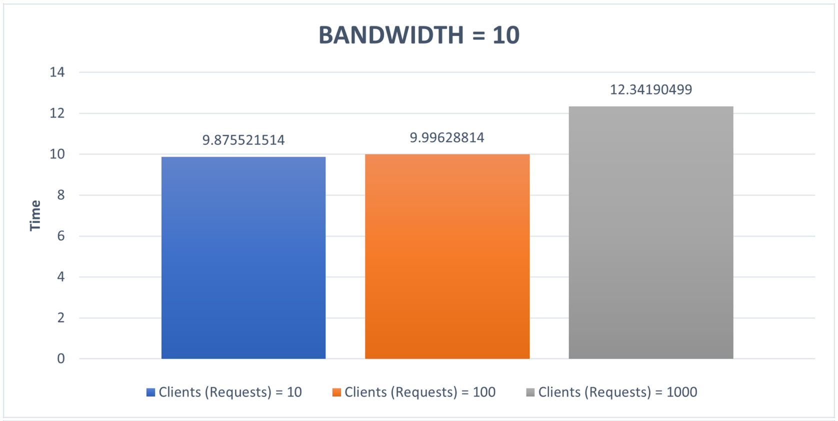
In this section of our report we experiment with different number of requests, data item variation and downstream bandwidth which are treated as inputs. The main point of this input variation is to observe how the average access latency is affected by the given inputs. In order make the charts more readable the two out of the three inputs remain the same and the third one changes. This process must be repeated for each different input set. A note to be made here is that the server waits for a computed number of time-slots before sending data to downstream. This delay helps us simulate the actual downstream bandwidth latency. After waiting, all responses are send to the downstream at once. In a more realistic implementation however, requests would be pipe-lined and the response time would actually be a little shorter.

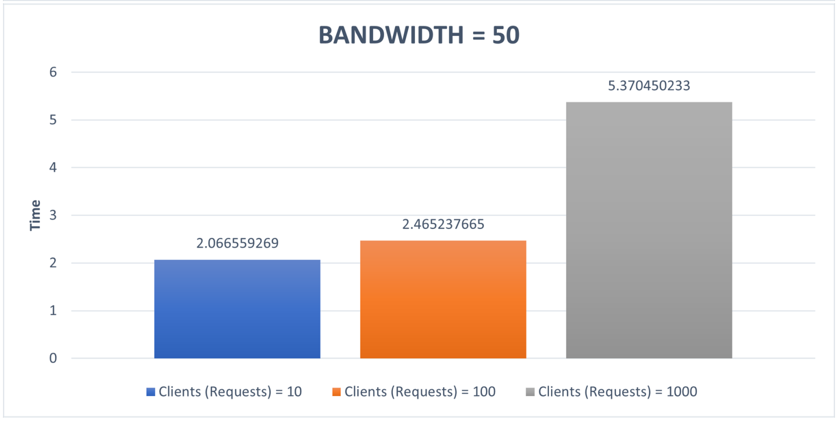
8.1 Increasing Data Variety

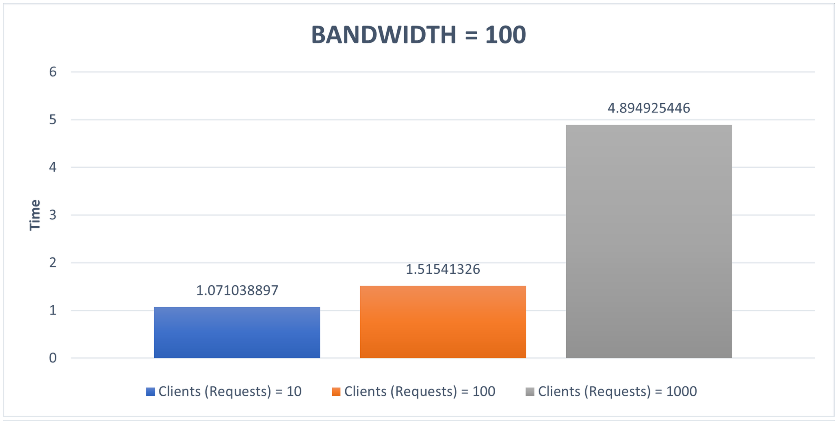
When more variety is added on the spawned data items the average access latency is rarely affected. This can be attributed to the fact that the scheduler does not depend on the data variety of the data items. The latency is only affected when a lot of clients wait for the same item which rarely happens when the variety of data items is increased. The chart for this scenario is not displayed since no important observations can be made. For the benchmarks bellow the total number of data items available is explicitly defined in the code (Data Variety = 100 Different Data Items).

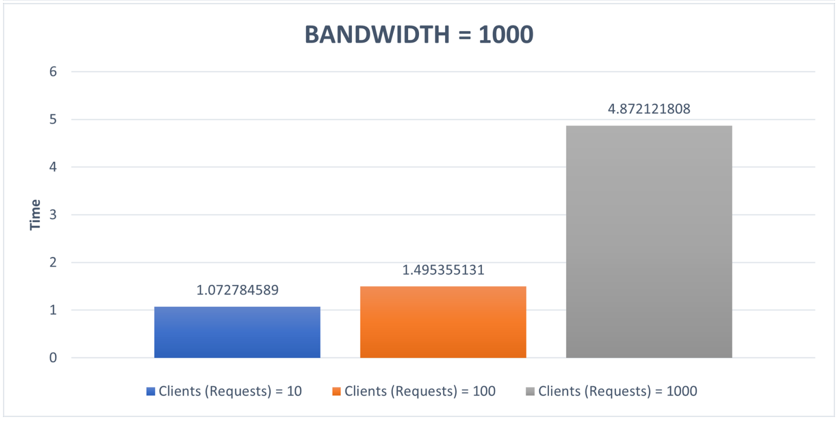
8.2 Increasing the Clients (Requests)

When more requests are submitted and neither the variety of data items nor bandwidth changes the average access latency is increased. This can be attributed to both the complexities of the MTRS which uses PuLP to solve the optimal request problem and MLRO which solves for optimal schedule. A chart of how the total number of clients and thus the total requests has a negative effect on the average access latency can be seen bellow.







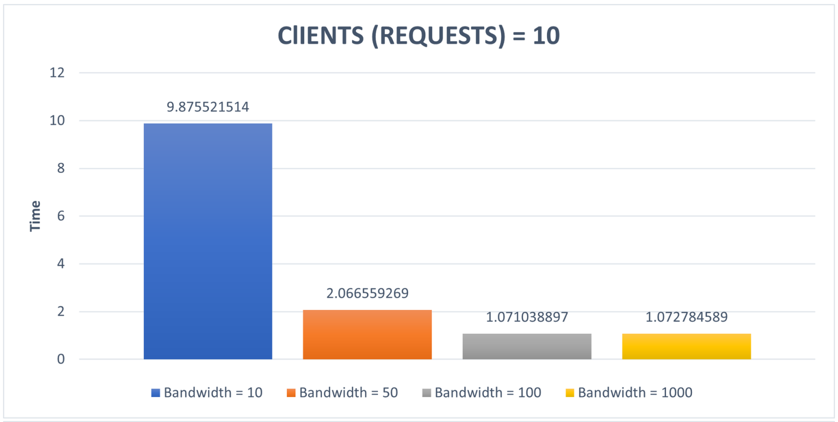


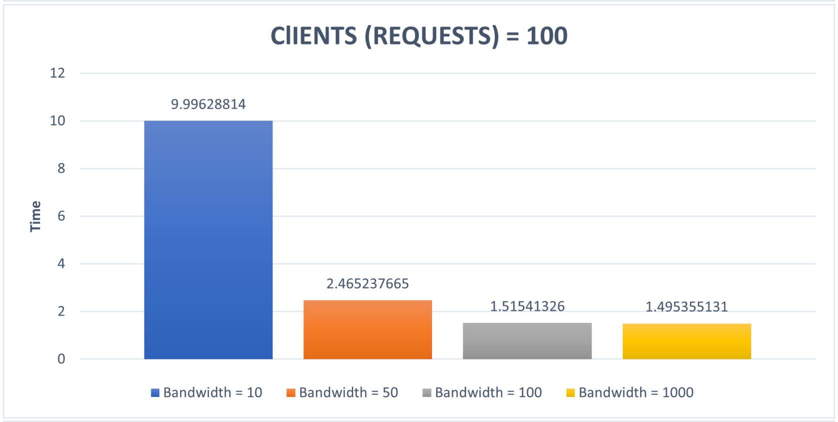
As it can be observed when more requests are made and the bandwidth is relatively small the average access latency is significantly increased. The scheduler is not at fault here, since it always tries to take advantage of all of the available bandwidth. The actual problem here actually is caused by the small bandwidth. More requests take a lot more time to be served since the system is bottle-necked by the downstream.

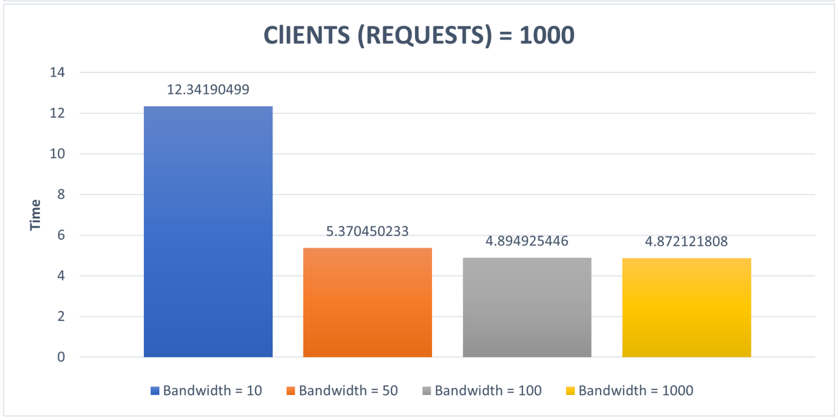
The increase of the average access latency may seem exponential at first, but it is heavily dependent on the downstream's bandwidth. After a certain number of requests the function of average access latency turns from exponential to logarithmic and finally converges. The point that the average access latency converges depends on the bandwidth.

8.3 Increasing the Bandwidth

When performing a pruning algorithm on the output of the MTRS the response size won't overflow the downstream and thus the response can be send in a certain amount of time slots given a certain bandwidth. The pruning algorithm is directly affected by the δ and thus by the bandwidth. When the bandwidth is increased the pruning algorithm is performed fewer times and more responses are sent to the downstream. A chart of how the bandwidth affects the average access latency, given that the total amount of clients remains the same, can be seen bellow.



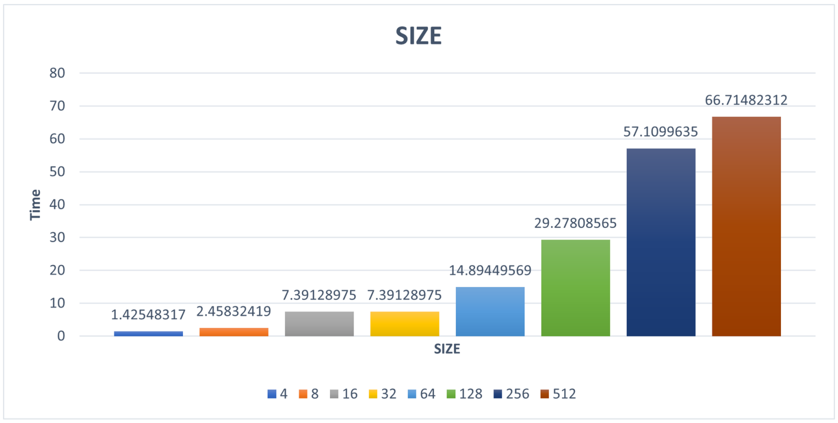




As it can be observed the average access latency is logarithmically reduced when the bandwidth is increased. The lower limit of average access latency is directly related to the MTRS, MLRO and Least Lost Heuristic algorithms time complexities.

8.4 Increasing the Size of Data Items

Our final experiment is changing the data item size in order to understand how the average access latency is affected. Given an explicit number of clients, server bandwidth and data item variety we increase the size of each data item and the results can be observed below.



As it can be observed the average access latency increases gradually as the data item size and thus the request size increases. The average access latency, at some point, will finally converge just like the paper suggests.

9 Contact Us

1. Στολτίδης Αλέξανδρος (2824): stalexandros@uth.gr
2. Λάντζος Στέργιος (2789): lstergios@uth.gr