Comparing Push- and Pull-Based Broadcasting
Or: Would "Microsoft Watches" Profit from a Transmitter?

Alexander Hall[1,⋆] and Hanjo Täubig[2,⋆⋆]

[1] Computer Engineering and Networks Laboratory, ETH Zürich,
CH-8092 Zürich, Switzerland, hall@tik.ee.ethz.ch
[2] Department of Computer Science, TU München,
D-85748 Garching b. München, Germany, taeubig@in.tum.de

Abstract. The first main goal of this paper is to present Sketch-it!, a framework aiming to facilitate development and experimental evaluation of new scheduling algorithms. It comprises many helpful data-structures, a graphical interface with several components and a library with implementations of selected scheduling algorithms. Every scheduling problem covered by the classification-scheme originally proposed by Graham et al. [22] can easily be integrated into the framework. One of the more recent enhancements of this scheme, the so called *broadcast scheduling* problem, was chosen for an extensive case study of Sketch-it!, yielding very interesting experimental results that represent the second main contribution of this paper. In broadcast scheduling many clients listen to a high bandwidth channel on which a server can transmit documents of a given set. Over time the clients request certain documents. In the *pull-based* setting each client has access to a slow bandwidth channel whereon it notifies the server about its requests. In the *push-based* setting no such channel exists. Instead it is assumed that requests for certain documents arrive randomly with probabilities known to the server. The goal in both settings is to generate broadcast schedules for these documents which minimize the average time a client has to wait until a request is answered. We conduct experiments with several algorithms on generated data. We distinguish scenarios for which a slow feedback channel is very advantageous, and others where its benefits are negligible, answering the question posed in the title.

1 Introduction

During the past 40 years scheduling problems have received a lot of research interest, a huge theoretical background was developed. For books on the topic see for instance [13–15]. While working on scheduling problems it would often be convenient to have a tool with which an algorithm could be implemented and tested quickly. "Playing" with the algorithm can help gaining intuition of how it works, or on the other hand potentially speed up the finding of counter-examples and bad cases. Quite often it is also meaningful to get hints on the performance of an algorithm by having a quick glance at

⋆ supported by the joint Berlin/Zürich graduate program CGC, financed by ETH Zürich and German Research Foundation (DFG)

⋆⋆ supported by DFG grant Ma 870/6-1 (SPP 1126: Algorithmics of large and complex Networks) and by DFG grant Ma 870/5-1 (Leibniz Award Ernst W. Mayr).

its empirical behavior. Then again some heuristics can only be evaluated by conducting such experiments. But besides for testing new algorithms, a tool which is able to animate the progress of an algorithm could also prove very helpful for presentations or in teaching topics of scheduling theory.

These points stimulated the development of Sketch-it!, a framework for simulation of scheduling algorithms. To maximize the applicability, its design was closely linked to the $\alpha|\beta|\gamma$-classification-scheme, originally proposed by Graham et al. [22]. Basically all problems covered by this scheme can be tackled with the help of the framework.

In this paper we give a short overview of the framework, in order to introduce it to a broad audience. Furthermore we present experimental results in the broadcast scheduling domain, which were obtained with the help of Sketch-it!. The motivation for this is partly to demonstrate the usability of the tool, but mainly we believe that the results are of interest in their own. In the next section we motivate and define broadcast scheduling.

Motivation and Problem Statement of Broadcast Scheduling Due to the increasing availability of infrastructure that supports high-bandwidth broadcast and due to the growth of information-centric applications, broadcast scheduling is gaining practical importance [4]. The general setting of the broadcast scheduling problem is that (possibly many) clients request documents (e.g. web pages) from a server, and the server answers these requests via a high-bandwidth channel to which all clients are connected. If several clients have requested the same document, a single broadcast of this document satisfies all their requests simultaneously. One wants to determine a broadcast schedule that optimizes some objective function, usually the average response time (the time a client has to wait on average until her request is satisfied; in the scheduling literature, this is also called the average flow time).

There are two principally different settings: on the one hand *on-demand* or *pull-based* broadcasts and on the other hand *push-based* broadcasts.

In the *pull-based* setting each client has access to a low-bandwidth *feedback* channel, e.g. a modem connection, whereon it notifies the server about its requests. Two examples of such systems are @Home network [24] and DirecPC [18], which provide Internet access via cable television and via satellite, respectively.

In the *push-based* setting no feedback channel exists. The server tries to anticipate user behavior from the previously observed popularities of individual documents. A classical example are Teletext systems, where the user can select a page on her remote control and then has to wait until it appears in the periodic broadcast. A more recent application is the SPOT technology announced by Microsoft [34]. It enables special wristwatches (and in the future also other devices) to receive personalized information—like weather, events and personal messages—via a dedicated radio frequency, which is shared for all broadcasts. The watches contain no transmitter, i.e. it is not possible to add a feedback channel. Clients configure their desired contents beforehand via a Web interface.

In this paper we present the first direct comparison of the empirical performance of several well known pull-based on-line algorithms with a push-based algorithm on the same input traces. Such a comparison may help, e.g., in deciding whether or not to integrate feedback channels into a broadcast system.

We now give a more precise problem definition. In the following we restrict ourselves to the *single channel* case (at any point in time no more than one document can be broadcasted), in the literature the case of $W > 1$ channels is also considered. Furthermore we allow *arbitrary-length* documents and *preemption*, i.e. a broadcast of a document can be interrupted and continued at a later point in time. We adopt the commonly made assumption that a client can *buffer* the last portion of a document and thus can start receiving the requested document immediately at any point of it. A request is satisfied as soon as all "units" of the document were received.

Let m be the number of documents and n be the total number of requests. By $l_i \in \mathbb{N}$ we denote the length of document $i \in \{1 \ldots m\}$. R_j, where $j \in \{1 \ldots n\}$, is used both to address the j-th request and to denote its arrival time. Let D_j be the document requested by R_j and $T = R_n$ be the arrival time of the last request.

The output of an algorithm in both the pull- and the push-based setting is a schedule $S(t) \in \{0 \ldots m\}$, for $t \in \mathbb{R}^+$, giving which document is broadcasted at time t, where $S(t) = 0$ means that no document is broadcasted. For simplicity we define $S^i(t) := 1$, if $S(t) = i$, and $S^i(t) := 0$ otherwise. Let T' be the point in time when all requests R_j are answered by S, w.l.o.g. $S(t) = 0$, for $t > T'$.

Input and objective function in the two settings differ though.

Pull-Based Here the total average response time ART is given by $\frac{1}{n}\sum_{j=1}^{n} F_j$, where $F_j := C_j - R_j$ is the response/flow time of request j and C_j is its completion time. More precisely $C_j = C^{D_j}(R_j)$, with: $C^i(t) := \inf\{x | \int_t^x S^i(\theta)d\theta = l_i\}$.

$B1|r_j, pmtn|\frac{1}{n}\sum F_j$ denotes the problem of minimizing the ART in this setting. An on-line algorithm for this problem only knows of the requests with $R_j < t$, when deciding which document to broadcast at time t. It is called ρ-competitive, if it computes a schedule whose ART is at most ρ times the ART of an optimal solution.

A W-speed ρ-competitive algorithm computes a schedule on W channels whose ART is at most ρ times the ART of an optimal solution on *one* channel.

Push-Based Unlike in the pull-based setting, the algorithm does not learn the actual requests R_j. Instead it is assumed that an infinite sequence of requests is generated in a Poisson process, i.e. the request interarrival times are exponentially distributed. The algorithm knows in advance the probabilities π_i with which a request R_j, $j \in \{1 \ldots n\}$ is for document $i \in \{1 \ldots m\}$. It computes an infinite (periodic) schedule which minimizes the *expected* instead of the average response time: ERT := $\mathbb{E}(\lim_{n\to\infty} \frac{1}{n}\sum_{j=1}^{n} F_j) = \mathbb{E}(F)$, where $\mathbb{E}(F)$ is the expected flow time of any request, given the Poisson assumption and the request probabilities π_i. The problem of minimizing the ERT is denoted by $B1|r_j, pmtn|\mathbb{E}(F)$. Note that algorithms are analyzed in an average case fashion, the output is usually not compared to an optimal solution for a given input.

An algorithm for this problem is called ρ-approximation, if it runs in polynomial time and computes a schedule whose ERT is at most ρ times the optimum ERT.

For our experiments we naturally only consider a finite sequence of requests R_j, $j \in \{1 \ldots n\}$. We limit the schedule $S(t)$ computed by a push-based algorithm to the interval $[0, T']$, where T' is the point in time when all requests R_j are answered by S, and as already mentioned above we assume $S(t) = 0$, for $t > T'$. We assume that the algorithm is fair and does not starve certain documents (i.e. T' is finite).

To facilitate the comparison of the results in the different settings, we will also compute the ART in the push-based case. Also in both settings let MRT denote the *maximum response time* of a request.

Organization of this paper and Contributions In the next section we give an overview of related work. Section 2 contains a high level description of the Sketch-it! framework. Finally in Section 3 we present our experimental results, which assert the selected push-based algorithm comparatively good performance with respect to ART, independently of how many documents are present. Furthermore the quality of the schedules do not vary depending on the document sizes. Contrasting this, the MRT comparatively increases with increasing m. A nice feature of the push-based algorithm is that it is independent of the load of a system, which means it scales nicely.

Related Work

Simulation A project similar to Sketch-it! is being developed since 1999 at the Institute for Algebra and Geometry of the Otto-von-Guericke University in Magdeburg. Its name *LiSA* stands for Library of Scheduling Algorithms, but actually it is specialized in shop scheduling. It is implemented in C++. For detailed documentation see [25].

The *Cheddar Project* is a simulation tool for real time scheduling, which was developed by the LIMI/EA 2215 team at the University of Brest (France) using the Ada programming language. It comes with a graphical editor and a library of classical real time scheduling algorithms and feasibility tests. For documentation and download of the distribution see [33].

Pull-Based Broadcast In [5, 6] Aksoy and Franklin introduce pull-based broadcast scheduling and conduct experiments with a proposed on-line algorithm for the case of unit-length documents and without preemption ($B1|r_i, p_i = 1|\frac{1}{n}\sum F_j$). Kalyanasundaram et al. [26], Erlebach and Hall [20] and Gandhi et al. [21] consider approximation algorithms for the corresponding off-line case. In [20] NP-hardness of the off-line setting is proved (also for $B1|r_i, pmtn|\frac{1}{n}\sum F_j$ with or without client buffering).

Acharya and Muthukrishnan [4] adopt the stretch-metric from [12] to broadcast scheduling and present experiments conducted with new on-line algorithms for arbitrary-length documents. Among other things, they show that preemption is very beneficial.

Edmonds and Pruhs [19] consider the case of arbitrary-length documents, with preemption ($B1|r_i, pmtn|\frac{1}{n}\sum F_j$) and *without* client buffering. They prove that no 1-speed $\mathrm{o}(n^{1/2})$-competitive algorithm exists (this result extends to the case with client buffering). They furthermore present the first positive result concerning the on-line setting: an $\mathcal{O}(1)$-speed $\mathcal{O}(1)$-competitive algorithm.

In [11], $B1|r_i, pmtn|\max F_j$ is addressed. They present a PTAS (polynomial-time approximation scheme) for the off-line case and show that First-Come-First-Serve is a 2-competitive on-line algorithm.

Push-Based Broadcast Ammar and Wong [7, 8] study the case of unit-length messages broadcasted on one channel ($B1|r_i, p_i = 1|\mathbb{E}(F)$) in the context of Teletext systems. For arbitrary number of channels W, this setting is also known as broadcast disks, treated e.g. in [1, 2]. There are several results for the variation of the problem where each complete broadcast of document $i \in \{1 \dots m\}$ is additionally assigned a cost c_i. The goal is to minimize the sum of the ERT and the *average broadcast cost*. Bar-Noy et al. [9] prove this is NP-hard for arbitrary c_i. They furthermore give a $9/8$- and a 1.57-approximation, if $c_i = 0$ respectively c_i arbitrary, for $i \in \{1 \dots m\}$. This is improved by Kenyon et al. [28]: they obtain a PTAS if the number W of channels and the costs c_i are bounded by constants.

For the case of arbitrary-length documents, without preemption see for instance [23, 38, 27]. For the case of arbitrary-length documents, with preemption Schabanel [36] presents a 2-approximation for $BW|r_i, c_i, pmtn|\mathbb{E}(F)$ based on the approach in [9] and adapts an NP-hardness proof from [27].

Acharya et al. [3] do an experimental study on interleaving push- and pull-based data broadcast for the case of unit length documents.

2 Sketch-it! – A new Scheduling Simulation Framework

The simulation framework was born in spring of 1999. It was developed under the supervision of Ernst W. Mayr, within the scope of the special research program SFB 342 of the German Research Foundation (DFG), part A7: "Efficient Parallel Algorithms and Schedules". A distribution version will be available soon at [32]. Source-code documentation can be found at [31].

Aim The original aim of the tool was to support research in the subject of scheduling that was done by the Efficient Algorithms group at the computer science department of the TUM. Such a tool should allow easy implementation of scheduling algorithms that are given by verbal description or pseudo code, and it should be flexible and open to all of the many different kinds of scheduling problems, each of which has its own special parameter settings and limitations. The goal was to prevent the user from doing the unnecessary work that always has to be done for the simulation, logging and visualization overhead. This suggested the name Sketch-it! (which was of course also chosen because of the phonetic proximity to the term 'schedule'). The researcher should be given a possibility to experiment with his own algorithms and ideas. This involves not only the implementation of algorithms, but also the creation of particular problem instances.

Another point of interest is the application of Sketch-it! as a teaching aid. This will be tested in an advanced seminar in the summer term of 2003.

Development Tools and Libraries The simulation framework is being developed in C++, using the standard compiler g++, provided by the GNU project. Since the complexity of the considered algorithms is of prime importance, we utilize the Library of Efficient Data types and Algorithms (LEDA) [30]. The visualization part of the application is implemented with Qt [35], a portable C++ development toolkit that mainly provides functionality for graphical user interfaces (GUI).

Simulation Core The core of the simulator comprises C++ classes for the different job types (single machine jobs, parallel jobs, malleable jobs, shop jobs, broadcast requests), classes for the environments (single machine, identical / uniform / unrelated parallel machines, flow / job / open shop), classes for the network topologies (set, line, mesh, hypercube) and classes to represent precedence constraints.

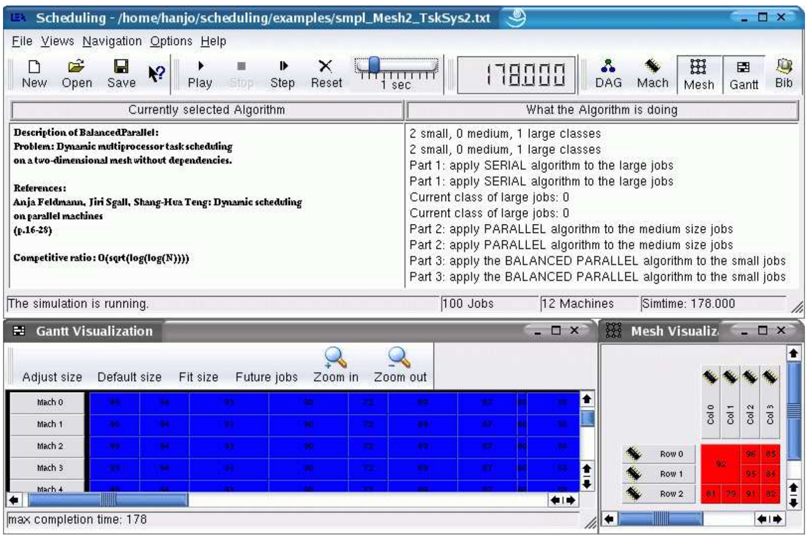
For a specific instance of a scheduling problem its jobs, machines and potentially precedence constraints are combined by a central class to form the *task-system*.

In the following section we describe briefly how the user can generate such a task-system and how a newly implemented or already existing algorithm can be executed with this task-system as input.

Usage

Problem Instance Generator To support testing and empirical analysis of algorithms instances can be automatically generated, one only needs to provide the parameters (environment, number of machines, number of jobs, probability distribution and expectation / variance values for stochastic problems, etc.). The tool then creates an instance of the requested problem and makes the appropriate random decisions (where necessary).

Simulation After selecting an appropriate scheduling algorithm (see below for a list of provided algorithms and how to implement a new one) the user can follow its execution either step-by-step or continuously at a certain speed. A Gantt chart which is constantly updated, shows the temporal assignment of jobs to machines. If precedence constraints are present, these can be displayed. Hereby jobs are highlighted according to their present state (i.e. released, available with respect to precedence constraints, running, finished). Other views can show the status of the machines (either in list form or in a special way, e.g. for a mesh topology). The main window also contains textual comments concerning the execution of the algorithm, e.g. about its current phase or about certain operations it is performing. A screenshot of Sketch-it!:



Logging Primarily to provide a standardized way to add new types of objective functions and also to ease the collection of relevant data during the execution of an algorithm, some logging capabilities are integrated in the form of so called loggers. These loggers are triggered by a carefully subdivided event system, which enables them to be notified in case selected events occur (like release of a new job, allocation of an available job, preemption of a running job, completion of a job, etc.). All standard objective functions (e.g. makespan, sum of completion times, sum of flow times, etc.) are already implemented with the help of such a logger.

Algorithms In order to implement a new scheduling algorithm, code needs to be inserted in dedicated functions for the “startup” and the “inner loop”. The “startup” function is meant for initialization purposes and is called before the actual execution. The “inner loop” function is called by Sketch-it! during the execution in an on-line fashion, every time a new job/request is available/released or a job/request was completed. Note that in case the algorithm needs to make decisions at other points in time, it can return a delay at the end of the function to notify Sketch-it! when it should to be called again. Off-line algorithms have complete access to the task-system and can compute the schedule already in the “startup” function.

To give a feeling of the spectrum of possible problems which can be attacked with Sketch-it! an incomplete list of algorithms implemented to date follows: Smith’s ratio rule for single machine scheduling, the algorithms of Coffman and Graham, the parallel machine algorithm of McNaughton for scheduling with preemptions, genetic algorithms for scheduling in an open shop environment, an algorithm of Ludwig and Tiwari for scheduling malleable parallel tasks, algorithms for stochastic scheduling (LEPT, SEPT, Smith weighted SEPT) and algorithms for the broadcast scheduling problems considered here. For descriptions of the multiprocessor task algorithms and of the algorithms for stochastic scheduling problems see [37] and [29].

3 Comparing Push- and Pull-Based Data-Broadcasting

In this section we present experiments which were conducted in order to compare the performance of push- to pull-based algorithms, with respect to the average response time (ART) and the maximum response time (MRT). Hereby the ART is emphasized, because it is the more widely accepted metric in the literature, and because a “good behavior” on the average seems more desirable. We compare several on-line algorithms for the pull-based setting to a push-based algorithm which is proven to be a 2-approximation for the expected response time (ERT). Note that the latter only gets the document probabilities π_i as input.

In the experiments document sizes l_i for $i \in \{1 \dots m\}$ and requests R_j, D_j for $j \in \{1 \dots n\}$, are generated at random. The ART/MRT of the computed schedules can vary considerably from one of these instances to the other. In order to obtain experimental data which can be compared easily for the different runs, simple lower bounds for the ART/MRT are calculated for each instance and the ART/MRT of the individual schedules are normalized by dividing with the respective lower bound. These lower bounds are described in the next section. In Section 3.2 we present the scheduling algorithms which were chosen to compete against each other. Then, in Section 3.3 we go into the details of how the instances are generated and finally discuss the results.

3.1 Simple lower bounds

The maximum of all message lengths is clearly a lower bound for the MRT. Let LM := $\max\{l_i \mid i \in \{1 \ldots m\}, D_j = i$ for some $j \in \{1 \ldots n\}\}$ denote this bound.

Deriving a lower bound LA for the ART is more interesting. First note that each request R_j, $j \in \{1 \ldots n\}$ obviously has to wait at least l_{D_j} units of time and thus LA $\geq \frac{1}{n}\sum_{j=1}^{n} l_{D_j}$.

To strengthen this trivial lower bound we consider points in time for which we know that at least one document is being broadcasted and others have to wait. To this end we define $w_i(t) := 1$, if there is a $j \in \{1 \ldots n\}$ with $D_j = i$ and $R_j \in (t - l_i, t]$, and $w_i(t) := 0$ otherwise, where $t \in [0, T_m]$ with $T_m := T + \max\{l_1 \ldots l_m\}$. In other words if $w_i(t) = 1$, there surely is at least one pending request for document i at time t. At any point in time $t \in [0, T_m]$ there are pending requests for at least $k(t) := \sum_{i=1}^{m} w_i(t)$ individual documents. Only one of these documents can be broadcasted at t and the $\geq \max\{k(t) - 1, 0\}$ requests for other documents must wait. This gives our lower bound: LA := $\frac{1}{n}\sum_{j=1}^{n} l_{D_j} \; + \; \frac{1}{n}\int_0^{T_m} \max\{k(t) - 1, 0\}dt$.

3.2 Implemented Algorithms

Pull-Based The first two are standard greedy on-line algorithms, known from traditional scheduling problems. The third was introduced in [4] and the fourth in [19].

• SSTF *Shortest Service Time First*: Like the name suggests, as time advances, always broadcast the document which has a pending request that can be satisfied most quickly among all pending requests. Note that a broadcast of a long document might be preempted if a request for a short document arrives. This on-line algorithm is optimal for the equivalent standard (unicast) scheduling problem $1|r_j, pmtn|\frac{1}{n}\sum F_j$, where it is known as *shortest remaining processing time*.

To show how easy it is to implement a scheduling algorithm with Sketch-it!, to the right we exemplarily present the actual source code inserted in the “inner loop” (cf. Section 2) for SSTF.

```
list<SBroadcastRequ*> a = getActiveRequ();
SBroadcastRequ *r, *p = NULL;
double s = MAXDOUBLE;
forall (r, a)
   if ( s > r→getRemProcTime() ) {
      s = r→getRemProcTime();
      p = r; }
if ( p != NULL )
   broadcastMsg( p→getMsgIndex() );
```

Experiments with SSTF in [4] imply good performance with respect to ART and bad performance with respect to MRT.

• LWF *Longest Wait First*: As time advances, always broadcast the document with the currently maximum total waiting time, i.e. the maximum total time that all pending requests for any document are waiting. In [26] it was conjectured that LWF is an $\mathcal{O}(1)$-speed $\mathcal{O}(1)$-competitive algorithm. Empirically it performs comparatively well with respect to MRT and not so well with respect to ART [4].

• LTSF *Longest Total Stretch First*: The current stretch of a request R_j at time $t \in [0, T']$ is the ratio of the time the request has been in the system so far $t - R_j$ (if it is still pending) respectively its response time F_j (if it is completely serviced) to the length of the requested document l_{D_j}. As time advances, LTSF always chooses to broadcast the document which has the largest total current stretch, considering all pending requests. It performs well empirically for ART; performance with respect to MRT varies [4].

• BEQUI-EDF *Equi-partition–Earliest Deadline First*: We only give a brief description of the algorithm, for details we refer to [19]. BEQUI-EDF runs in two stages, in the first stage Equi-partition is simulated: the broadcast bandwidth is distributed among the documents which have pending requests, where each document receives bandwidth proportional to the number of unsatisfied requests for it. Each time a document was completely broadcasted in this simulated first stage of the algorithm, a deadline is derived from the current time and the time it took to broadcast the document. In the second stage at any time the document with the earliest deadline is broadcasted.

This algorithm is special, because it is so far the only algorithm with a proven worst case performance. In [19] it is shown for any $\varepsilon > 0$ to be $(1 + \varepsilon)(4 + \varepsilon)$-speed $\mathcal{O}(1)$-competitive, if clients cannot buffer and where the constant $\mathcal{O}(1)$ depends on ε. In other words if we e.g. set $\varepsilon = 0.1$, the ART of a schedule computed by BEQUI-EDF for one channel is at most $\mathcal{O}(1)$ times the ART of an optimal schedule which only has access to a channel with a bandwidth of ≈ 0.22. Note that it is impossible to obtain a 1-speed competitive algorithm with ratio better than $\mathrm{o}(n^{1/2})$, see Section 1.

Unfortunately the competitive analysis of BEQUI-EDF does not carry over to our model, where client buffering is enabled. We nevertheless think it is of interest to see how well an algorithm with provable worst case behavior (although for a slightly different model) performs compared to commonly used heuristics.

Push-Based Most results in the literature are concerned with the unit-length case, although there is some work on arbitrary-length messages, without preemption, see Section 1. The algorithm proposed in [36] (which is a 2-approximation with respect to the ERT) is the only candidate to date for the push-based setting with arbitrary-length messages and preemption.

• PUSH: The algorithm expects the probabilities π_i as input, we simply estimate these from the given requests R_j, i.e. we set $\pi_i = \frac{1}{n}|\{j \mid j \in \{1 \dots n\}, D_j = i\}|$. For randomly generated instances one has direct access to the underlying probabilities, see also next section.

The documents are split into pages of length 1 (this is possible because $l_i \in \mathbb{N}$ for $i \in \{1 \dots m\}$) and in each time-step a document is selected and its next page in cyclic order is broadcasted.

Schabanel [36] considers the case where broadcast costs c_i may be present. If these are set to zero, the analysis can be considerably simplified and yields that the pages of document $i \in \{1 \dots m\}$ should be broadcasted such that the expected distance between two consecutive broadcasts of i is $\tau_i := \mu / \sqrt{\pi_i l_i}$, where $\mu = \sum_{i=1}^{m} \sqrt{\pi_i l_i}$ is a normalizing factor. These expected distances can be achieved by a simple randomized algorithm: at each time-step choose to broadcast document $i \in \{1 \dots m\}$ with probability $1/\tau_i$. Note that $\sum_{i=1}^{m} 1/\tau_i = 1$. We implemented the derandomized greedy algorithm also given in [36].

3.3 The Experiments

Generation of instances To be able to sample instances for broadcast scheduling problems, Sketch-it!'s generator needed to be enhanced in a straightforward way. We chose to generate the instances such that the interarrival times of requests are exponentially distributed with rate λ. This is a natural assumption which is often made in such a context, e.g. in queuing theory. For each request R_j, $j \in \{1 \ldots n\}$ we choose $D_j = i \in \{1 \ldots m\}$ at random with probability π_i. To assess realistic values for the probabilities π_i, we oriented ourselves at document popularities in the Internet. These are widely believed to behave according to Zipf's Law, thus we choose

$$\pi_i \propto i^{-1},$$

assuming that index i corresponds directly to the rank of a document, i.e. $i = 1$ is the most popular document, $i = 2$ the second most, and so on. In [10] it is stated that an exponent of -1 models reality well. PUSH can directly be given these probabilities π_i as input. Experiments showed that it makes no perceivable difference for our setup, whether PUSH obtains the π_i or their estimates as described in the previous section (the following discussion is based on the case where the π_i are estimated).

It remains to choose a realistic distribution for the document sizes. [17] contains a comprehensive study on file sizes in the Internet. It yields that it is reasonable to assume file sizes to be Pareto distributed:

$$\Pr(\text{“file size”} > x) \propto x^{-\alpha},$$

with $x \geq k$ and $\alpha \in (0, 2]$, $k > 0$, whereby it is stated in [17] that $\alpha = 1.2$ is realistic. As to not get arbitrarily large documents we chose the bounded Pareto distribution (see e.g. [16]) for the interval $[1, 100]$. Furthermore we rounded the sizes to integer values.

Conducted Experiments Of many interesting questions, we selected two:

1. "How do the scheduling algorithms perform depending on the number of documents m?" To this end we ran tests, stepwise increasing the number of documents, starting from $m = 2$ up to $m = 150$. For each fixed m we generated 100 instances with 1000–4000 requests each, as described in the previous section. From this we calculate the mean and variance of $\frac{\text{ART}}{\text{LA}}$ respectively $\frac{\text{MRT}}{\text{LM}}$ of all 100 instances for each scheduling algorithm. The empirical variance shows how predictable a scheduling algorithm is. It often is better to choose a predictable algorithm (i.e. with low variance), even if on average it performs slightly worse than others. We set the request arrival rate to $\lambda = 2$, that is in expectation 2 requests arrive per unit of time.
2. "How do the algorithms perform depending on how heavily the system is loaded?" To simulate behavior with different loads, we set $\lambda = 2^b$ for $b \in \{-6 \ldots 4\}$, instead of varying m. Everything else is done as above, m is set to 20.

Discussion of the results We now discuss the plots in Figure 1 (appendix) row by row.
1. ART: Most strikingly, SSTF behaves very badly, PUSH behaves about the same as the others and BEQUI-EDF is second best. A closer examination of the individual algorithms follows.
• *Pull-Based*: SSTF shows the worst behavior of all algorithms. For all m it not only has the highest ART, but also a very high variance, i.e. the quality of a solution is very unpredictable and depends heavily on certain properties of the current input.

This outcome is quite surprising, because in [4] SSTF's overall empirical performance concerning ART is very good compared to LWF, LTSF and other algorithms.

A possible explanation for the observed behavior is that in some instances requests for short documents appear often enough to starve a significant number of requests for longer ones. In other words, constantly arriving requests for some short document i_1 might each time preempt the broadcast of a longer document i_2 such that more and more waiting requests for i_2 accumulate. This suggests that SSTF's performance greatly depends on how long popular documents are compared to slightly less popular ones. Note that SSTF is the only pull-based algorithm which in no way takes into account the number of requests waiting for the individual documents. This presumably explains its exceptional status in all plots.

LWF performs worst with the exception of SSTF. In [4] it has the overall worst performance among the tested algorithms with respect to the ART. LTSF empirically has the lowest ART, this confirms [4], where it also performs very well. For BEQUI-EDF the results look very promising. This is somewhat astonishing because the algorithm first simulates Equi-partition and then inserts deadlines at seemingly late points in time. A possible reason why it nevertheless performs well is given below.
• *Push-Based*: Also PUSH does comparatively well, which again is somewhat surprising: one could have expected a bigger advantage of the pull-based algorithms, especially when the number of documents in the system increases. This result is very interesting and shows that PUSH also empirically performs well on instances with exponentially distributed interarrival times. In particular it is very robust against variation of file sizes. Note: the sizes are varied independently of document popularities.
2. MRT: We do not go into details, just note that the good performance of LWF seems somehow intuitive: if requests with the longest waiting time are greedily selected, the probability that some request is starved for a long period of time is very small. On the other hand the MRT of PUSH continuously increases. This might stem from the fact that with increasing number of documents the smallest document probability π_m decreases. If such a document is requested, it might take a long time until it is completely broadcasted by PUSH.
3. ART: For the mentioned reasons SSFT is again by far worst for $\lambda > 1$. PUSH and BEQUI-EDF both are comparatively bad for low loads, but improve with higher loads. For $\lambda = 8, 16$ they even show the best performance of all algorithms.

It seems that the ART of PUSH stays at about the same level, independently of λ. This would not be surprising: if all π_i and l_i are fixed, PUSH computes a schedule which is independent of the choice of λ. Thus the expected response time is also constant (this time is obviously independent of λ for a fixed schedule). So PUSH simply benefits from a heavily loaded system because the pull-based algorithms cannot exploit their advantage of knowing which documents are currently actually requested. This they can do for lower loads, where PUSH also performs comparatively worse.

The ART of BEQUI-EDF is presumably so high in systems with low load, because for each request it has to wait quite long until it can set a corresponding deadline for EDF. The simulated Equi-partition algorithm divides the bandwidth according to the number of outstanding requests for the individual documents. Thus in systems with high loads this might implicitly give estimations of the document probabilities π_i for a certain time window. This could perhaps explain why it behaves so similarly to PUSH.
4. MRT: Except for SSTF, BEQUI-EDF performs worst for high λ. This is not astonishing because the algorithm is not at all trimmed to minimize the MRT. The MRT of PUSH again stays at about the same value, for the same reason as the ART does.

4 Conclusions

Sketch-it proved itself very useful while conducting the experiments. It was quite easy to implement the algorithms and also to enhance the generator and create the test suites.

From the experiments we conclude that the pull-based algorithm SSTF does not carry over well from traditional (unicast) scheduling. Furthermore the push-based algorithm PUSH is very robust for the case of exponentially distributed interarrival times. It performs well compared to the pull-based algorithms, independently of the number of documents and the distribution of file sizes among differently popular documents. On highly loaded systems it even outperforms pull-based algorithms, because they cannot exploit their advantage of knowing which documents the individual requests are actually for. These results are quite promising for Microsoft's SPOT technology, if one assumes that primarily information like weather and news is broadcasted (i.e. a reasonable number of documents, and the Poisson assumption is close to reality). In particular they are promising because it is probable that the system load is high: potentially a lot of users are listening to an extremely low-bandwidth channel. Moreover PULL's empirically observed independence of the load (which confirms theoretical results) makes this setting nicely scalable. On the other hand when the number of documents increases (e.g. if the possibility to send personal messages is added to the system) the ART is still reasonable (in the experiments about twice the amount of the best algorithm's output), but the MRT increases distinctly. This could mean that some users have to wait long for messages of low popularity (e.g. personal messages) and might get frustrated.

An interesting open question would be to find out how the algorithms perform on real world data (e.g. traces of Web-Servers) or generated data where the requests do not stem from a Poisson process.

Acknowledgments

The authors are much indebted to Ernst W. Mayr for stimulating the development of Sketch-it!. We would like to thank Thomas Schickinger, who played an important role in the original design and implementation of the tool. Furthermore we would like to thank Thomas Erlebach for numerous helpful discussions on the topic of broadcast scheduling and many appreciated comments concerning this paper.

References

1. S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communication environments. In *Proceedings of the ACM SIGMOD Conference*, pages 199–210, San Jose, CA, May 1995.
2. S. Acharya, M. Franklin, and S. Zdonik. Prefetching from a broadcast disk. In *Proceedings of the 12th Int. Conference on Data Engineering (ICDE)*, pages 276–285, New Orleans, Louisiana, February 1996.
3. S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull for data broadcast. In *Proceedings of the ACM SIGMOD*, Tuscon, Arizona, May 1997.
4. S. Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: New metrics and algorithms. In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 43–54, Dallas, Texas, USA, October 1998. ACM Press.
5. D. Aksoy and M. Franklin. Scheduling for large-scale on-demand data broadcasting. In *Proceedings of the IEEE INFOCOM Conference*, pages 651–659, San Francisco, CA, March 1998.
6. D. Aksoy and M. Franklin. RxW: A scheduling approach for large-scale on-demand data broadcast. *ACM/IEEE Transactions on Networking*, 7(6):846–860, December 1999.
7. M. H. Ammar and J. W. Wong. The design of teletext broadcast cycles. *Performance Evaluation*, 5(4):235–242, Dec 1985.
8. M. H. Ammar and J. W.Wong. On the optimality of cyclic transmission in teletext systems. *IEEE Transaction on Communication, COM*, 35(1):68–73, Jan 1987.
9. A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieber. Minimizing service and operation costs of periodic scheduling. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 11–20, San Francisco, California, January 1998. ACM Press.
10. P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of ACM SIGMETRICS, Measurement and Modeling of Computer Systems*, pages 151–160, Madison, Wisconsin, USA, June 1998. ACM Press.
11. Y. Bartal and S. Muthukrishnan. Minimizing maximum response time in scheduling broadcasts. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 558–559, San Francisco, California, January 2000. ACM Press.
12. M. A. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 270–279, San Francisco, California, January 1998. ACM Press.
13. J. Błażewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Węglarz. *Scheduling Computer and Manufacturing Processes*. Springer-Verlag, 1996.
14. P. Brucker. *Scheduling Algorithms*. Springer-Verlag, 3rd edition, September 2001.
15. P. Chrétienne, E. G. Coffman, Jr., J. K. Lenstra, and Z. Liu. *Scheduling Theory and its Applications*. John Wiley & Sons, 1995.
16. M. Crovella, M. Harchol-Balter, and C. D. Murta. Task assignment in a distributed system: Improving performance by unbalancing load. In *Proceedings of ACM SIGMETRICS, Measurement and Modeling of Computer Systems*, pages 268–269, Madison, Wisconsin, USA, June 1998. ACM Press.
17. M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.
18. DirecPC. Website. www.direcpc.com.
19. J. Edmonds and K. Pruhs. Broadcast scheduling: When fairness is fine. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 421–430, San Francisco, California, January 2002. ACM Press.

20. T. Erlebach and A. Hall. NP-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 194–202, San Francisco, California, January 2002. ACM Press.
21. R. Gandhi, S. Khuller, Y.-A. Kim, and Y.-C. Wan. Algorithms for minimizing response time in broadcast scheduling. In *Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, volume 2337 of *LNCS*, pages 425–438, Cambridge, MA, USA, May 2002. Springer-Verlag.
22. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Math*, 5:287–326, 1979.
23. S. Hameed and N. H. Vaidya. Efficient algorithms for scheduling data broadcast. *Wireless Networks*, 5(3):183–193, May 1999.
24. @Home Network. Website. www.home.net.
25. Institut für Algebra und Geometrie, Otto-von-Guericke-Universität Magdeburg. LiSA Homepage. lisa.math.uni-magdeburg.de.
26. B. Kalyanasundaram, K. Pruhs, and M. Velauthapillai. Scheduling broadcasts in wireless networks. In *Proc. of the 8th Annual European Symposium on Algorithms (ESA)*, volume 1879 of *LNCS*, pages 290–301, Saarbrücken, Germany, September 2000. Springer-Verlag.
27. C. Kenyon and N. Schabanel. The data broadcast problem with non-uniform transmission times. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 547–556, January 1999.
28. C. Kenyon, N. Schabanel, and N. E. Young. Polynomial-time approximation scheme for data broadcast. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 659–666, May 2000.
29. B. Meier. Simulation of algorithms for stochastic scheduling problems. Diploma thesis, Technische Universität München, February 2003.
30. Algorithmic Solutions Software GmbH. LEDA. www.algorithmic-solutions.com/.
31. Lehrstuhl für Effiziente Algorithmen. Sketch-it! Documentation. wwwmayr.informatik.tu-muenchen.de/scheduling/scheduling-api/html/classes.html.
32. Lehrstuhl für Effiziente Algorithmen. Sketch-it! Homepage. wwwmayr.informatik.tu-muenchen.de/scheduling/.
33. LIMI/EA 2215, University of Brest. The Cheddar project: a free real time scheduling simulator. http://beru.univ-brest.fr/ singhoff/cheddar/.
34. Microsoft PressPass. Microsoft Presents Smart Personal Objects Technology (SPOT)-Based Wristwatches at CES. www.microsoft.com/presspass/press/2003/jan03/01-09SPOTWatchesPR.asp.
35. TrollTech, Norway. Qt. www.trolltech.com/products/qt/.
36. N. Schabanel. The data broadcast problem with preemption. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1770 of *LNCS*, pages 181–192, Lille, France, February 2000. Springer-Verlag.
37. H. Täubig. Simulation of multiprocessor scheduling problems. Diploma thesis, Technische Universität München, August 2000.
38. N. H. Vaidya and S. Hameed. Scheduling data broadcast in asymmetric communication environments. *Wireless Networks*, 5(3):171–182, May 1999.

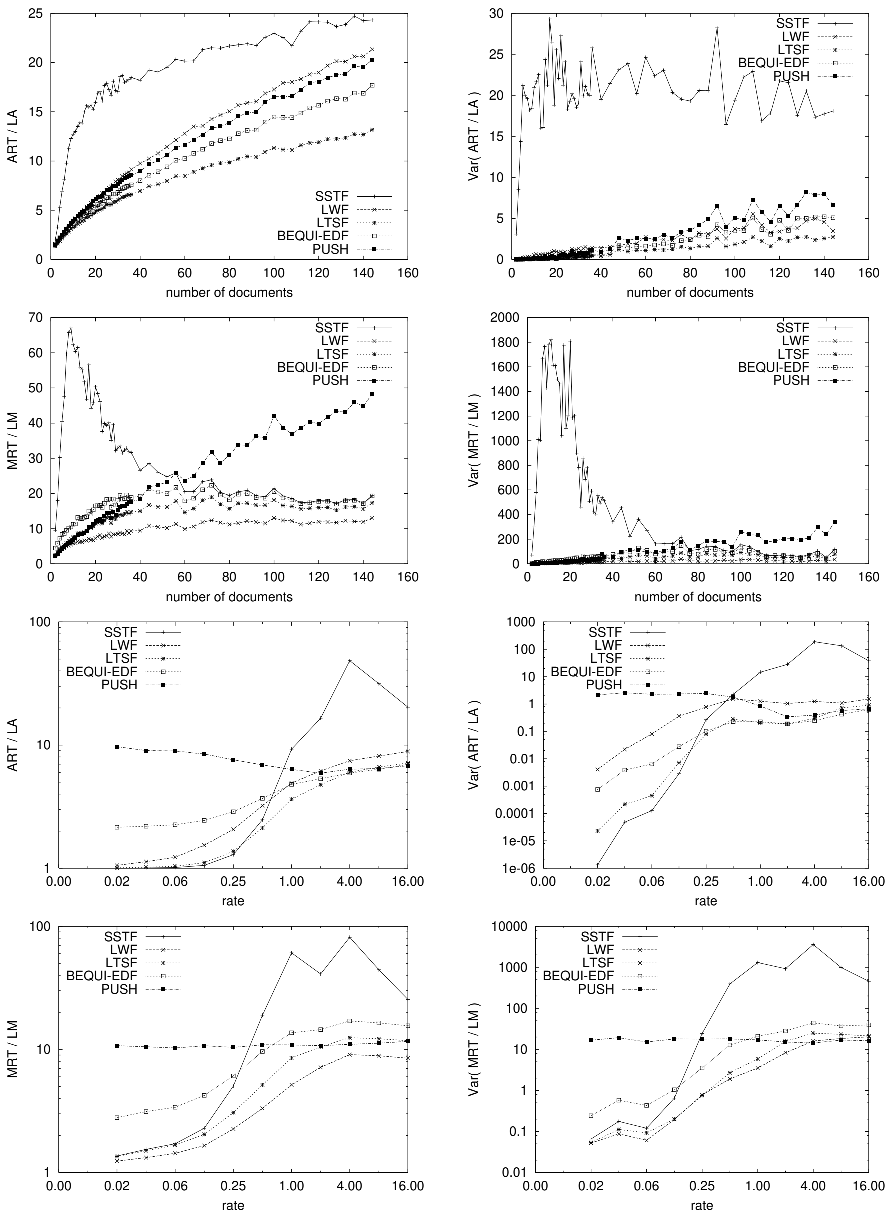


Fig. 1. Generated data. Plots of the ART, the variance of the ART, the MRT, and the variance of the MRT in dependency of the number of documents m respectively rate λ. Each quantity is normalized by the corresponding lower bound, see text for details. For each data point 100 input traces were generated, each containing 1000–4000 requests. For the top 4 plots we chose $\lambda = 2$, i.e. in expectation 2 request arrive in one unit of time. For the lower 4 plots we chose $m = 20$.