Utility-based Scheduling for Public Displays with Live Content

Kristi Bushman
University of Pittsburgh
Pittsburgh, PA
k.bushman@pitt.edu

Alexandros Labrinidis
University of Pittsburgh
Pittsburgh, PA
labrinid@cs.pitt.edu

ABSTRACT

The pervasiveness of public displays is prompting an increased need for "fresh" content to be shown, that is highly engaging and useful to passerbys. As such, live or time-sensitive content is often shown in conjunction with "traditional" static content, which creates scheduling challenges. In this work, we propose a utility-based framework and a novel scheduling algorithm for handling live and non-live content on public displays. We also experimentally evaluate our proposed algorithm against a number of alternatives under a variety of workloads.

CCS CONCEPTS

• **Human-centered computing → Human computer interaction (HCI)**.

KEYWORDS

pervasive displays, scheduling algorithm, utility function, deadlines

ACM Reference Format:
Kristi Bushman and Alexandros Labrinidis. 2019. Utility-based Scheduling for Public Displays with Live Content. In *Proceedings of the 8th ACM International Symposium on Pervasive Displays (PerDis '19), June 12–14, 2019, Palermo, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3321335.3324940

1 INTRODUCTION

Pervasive display networks are becoming a regular fixture of everyday city life [3]. Although the majority of such displays are still showing mostly static content, e.g., advertisements, the push and the demand for data-rich content is very high. Data-rich content is often *live* (e.g., real-time transit information[1]) or *time-sensitive* (e.g., weather forecasts). One way to address the idiosyncrasies of live/real-time content is to assign *deadlines* to it, i.e., a specific time point by which the content item should be displayed in order to have maximum positive "value" to passerbys. Of course, such deadline-driven content should coexist with content that does not have such specific timing requirements.

PerDis '19, June 12–14, 2019, Palermo, Italy

ACM ISBN 978-1-4503-6751-6/19/06…$15.00
https://doi.org/10.1145/3321335.3324940

1.1 Motivating Example

Our motivating example is a public display at a bus stop that shows various content items that include real-time bus arrival information, real-time traffic information, up to the minute weather information, the Twitter feed of the bus company, and advertisements. The motivation behind these content choices is to make the display "interesting," so that it does not get ignored like banner ads on web sites. Along those lines, we envision different content items being shown at separate times on the display, instead of trying to squeeze too many things in a single screen at the same time.

Given this setup, we want to determine the best schedule to show the various content items. Clearly, the different types of content have different "value" to the people at the bus stop and that value changes over time. For example, it is absolutely crucial that information about a bus arrival be shown shortly before the bus arrives (30 seconds - 1 minute) and definitely not after the bus leaves the bus stop. The exact arrival time of the bus (i.e., the "deadline") is often not the originally scheduled time, since it is affected by current traffic conditions, and therefore not known well in advance. Additionally, different content types (e.g., Twitter feeds) do not have such strict timing constraints, but must also be shown.

1.2 Limitations of the State of the Art

There are a plethora of scheduling algorithms [1, 6, 8, 15], including several algorithms specifically designed for determining the timing of content shown on public displays [3–5, 11–14, 17, 18]. We classify these algorithms into two categories, *procedural* and *declarative*, borrowing terminology from programming languages.

Procedural scheduling algorithms require the display owner to specify the exact schedule ahead of time. That could be either the exact times to show each content item or an order of the different content items (with durations for each) that is played in a constant loop or for specific time periods[2]. For content that has deadlines, the display owner would need to manually schedule it ahead of time, something that would clearly be a problem when the deadline is not known far in advance.

Declarative scheduling algorithms require the display owner to specify some notion of importance or rules for the different content items and it would then be up to the scheduling algorithm to determine the exact timings. These can be further subdivided into two classes, based on whether the scheduling algorithm decides the full schedule ahead of time [16] or each item just in time [11].

The state of the art in this space is the work by Mikusz et al [11] using lottery scheduling. The lottery scheduling algorithm allocates tickets to content items based on some scheduling policy, then randomly draws a ticket to decide which item to show. A change in

[1] https://transitscreen.com
[2] https://screen.cloud

scheduling priorities can be reflected by changing the allocation of lottery tickets. Although this technique is very flexible, it will not work well when the timing of when to show an item is absolutely crucial to its value to viewers, as is the case with our motivating example. Specifically, the algorithm has no notion of the future, so it is unable to recognize when it should hold off on showing an item in order to be more valuable. Additionally, the random nature of the algorithm opens up the possibility of missed deadlines.

1.3 Requirements of an Ideal Scheduling Algorithm

Given the motivating example described earlier, we would like to have a scheduling algorithm with the following characteristics:

- Can handle scheduling constraints (e.g., what time of the day to show specific content).
- Can deal with content items being added to or deleted from the list of available content (even without significant advance notice).
- Can handle content that is deadline-driven, but also content that is not.
- Can consider the different "value" content has to passerbys and use it to prioritize scheduling decisions.

To the best of our knowledge, there is no scheduling algorithm that addresses all the above characteristics.

1.4 Contributions of this Work

We make the following contributions:

(1) We highlight the need for deadline-driven scheduling of content for public displays.
(2) We design a content scheduler architecture to address the needs of deadline-driven and non deadline-driven content (Section 2).
(3) We propose a utility function framework to capture the inherent "value" for the different content items (Section 3).
(4) We develop a novel scheduling algorithm (Lookahead Algorithm) that tries to minimize missed deadlines while maximizing the overall utility (Section 4).
(5) We experimentally evaluate the proposed scheduling algorithm against multiple alternatives under a variety of workloads (Sections 5, 6).

2 SCHEDULER ARCHITECTURE

We envision a public display scheduler that consists of three components: (a) *a content library*, (b) *a filterer*, and (c) *a content scheduler*. The content library stores information about the content items. Content items can be added to or removed from the library at any time. This can be done through a user interface or programmatically through an API. Content items submitted to the library must include: content info (image, video, web URL, etc.), duration, valid days and times, and a utility function. The filterer component pulls content from the library and removes invalid items before passing them to the scheduler. A content item is invalid if the current time is not within valid times specified for the item. Finally, the content scheduler then decides which item to show out of the content items that were passed to it by the filterer.

3 PROPOSED UTILITY FUNCTION FRAMEWORK

Utility functions (UF) are used in many disciplines in order to express value over time. UFs that express the value of job completion over time have been used for scheduling tasks in real-time operating systems [7], database systems [9], and HPC systems [10].

In our framework, each content item has an associated utility function which represents the viewer-perceived value of showing the item over time. Content items can be partitioned into two categories: (a) **anytime content (AC)** and (b) **deadline-driven content (DC)**. We believe that any type of content can fit into this framework, and this framework will allow us to represent many common scheduling constraints.

An anytime content item has no inherent value tied to a specific time of day. However, it may increase in value to viewers if not shown for some period of time. An example of an AC item is a weather application. The weather is valuable to viewers at any time of day, however, it is not valuable to show the weather twice in one minute, as major updates to the forecast are unlikely. Other examples of AC items include news applications, Twitter feeds, and advertisements. Scheduling algorithms that cycle through a pre-determined playlist are only able to show AC items. There is no constraint on the number of times an AC item should be shown.

A deadline-driven content item is tied to a very specific time of day, which may not be known far in advance. Often, these content items will be related to live events, making the timing of when to show the item critical. DC items should only be shown once. An example of a DC item is an alert that says "Bus Arriving Now". Ideally, this item would be shown 30 seconds before the bus arrives. Showing this item too early or too late would cause confusion and provide no value to viewers. Other examples of DC items include emergency alerts, event reminders, and live video streams.

Note that the value of an AC item can be tied to a specific time of day, however, this will likely be a longer time frame for which the content is valid. For example, an advertisement for a breakfast cafe may only be valid between 7am and 11am. However, within that time frame, the advertisement can be shown multiple times and the value of showing the advertisement is dependent on the time since the advertisement was last shown.

The utility function of an AC item is a non-decreasing function, where the x-axis units are time offsets relative to the time when the content item was last shown. The y-axis units are the user-perceived value of showing the item. Immediately after an AC item is shown, the value of its utility function goes back to the value at time offset zero. The utility function of a DC item must increase from zero at some time (t_s) and return to zero at its deadline (t_d). The x-axis units for a DC utility function are absolute times of day. The y-axis units are user-perceived value. For both AC and DC items, the utility acquired by showing the item at a certain time is represented by the integral of the UF over the duration (d) that the content is shown.

4 LOOKAHEAD SCHEDULING ALGORITHM

In this section, we propose an algorithm called the Lookahead algorithm (LA) for scheduling content on public displays using our utility function framework. The goal of this algorithm is to decide

which content item to show next in order to maximize both the total utility of content shown and the number of DC items that are shown before their deadlines.

When deciding which content item to show next, at time t_n, we construct a lookahead window of size w (seconds). The lookahead window is a period of time where we will construct a hypothetical schedule of what is likely to be shown in the near future. This window helps inform our decision of which content item to show at time t_n. When constructing this hypothetical schedule, we only consider DC UF's that contain some value within the window. We will decide whether to show an AC item after construction of the hypothetical schedule is completed.

For all DC content within the window, we calculate the slack of the utility function (Eq. 1). Our definition of slack was inspired by the notion of slack in operating systems[3]. Slack measures how many time slots are available for scheduling the content item while also receiving utility value. Larger slack means there are more options for when to schedule that item. We place DC items on the hypothetical schedule in order of increasing slack (i.e. DC items with the least slack are placed first). Each content item is placed at the time where its acquired utility (integral of the UF) is maximized given that it does not conflict with any item already on the schedule. If there are multiple time slots that tie for the highest acquired utility, the content is placed in the earliest of those time slots. A content item is not placed on the hypothetical schedule if it cannot acquire any utility.

$$slack = \frac{t_d - \max(t_n, t_s)}{d} \tag{1}$$

Once all valid DC items have been placed on the hypothetical schedule, we look at the very beginning of the hypothetical schedule. If there is a content item placed on the hypothetical schedule at the very beginning, that is the content item that will be shown at time t_n. Otherwise, we calculate the gap of time from the beginning of the hypothetical schedule to the first DC item on the hypothetical schedule. Out of the AC items with a duration that would fit in this gap, the item with the highest utility density (Eq. 2) [7] for its duration is the item that will be shown at time t_n.

$$density = \frac{\int_{t_n}^{t_n+d} UF}{d} \tag{2}$$

This decision process is executed within the last second of showing the current content so that the decision of what to show next is based on the most current knowledge of upcoming content. For every decision made, a new lookahead window is completely reconstructed. Although content items are likely to be placed in the same time slot on the hypothetical schedule for many iterations of the decision process, recalculating the hypothetical schedule with every iteration allows the algorithm to be responsive to new content, while still using available knowledge to inform the current decision.

5 EXPERIMENTAL SETUP

5.1 Evaluation Environment

We implemented a simulator program in Python to evaluate different scheduling algorithms; it was executed on a Dell machine with an Intel Core i7 3.4 GHz processor and 32 GB of RAM.

5.2 Algorithms Evaluated

We evaluated the performance of seven different algorithms:

- EDF **Earliest Deadline First**: For any DC item that would acquire some utility if shown next, show the content item with the earliest deadline. If there are no such DC items, show the AC item with the highest utility density.
- G **Greedy**: Choose the content item with the highest utility density (as specified in Eq. 1).
- LA **Lookahead**: As described in Section 4. A lookahead window of 5 minutes was used in our evaluation.
- LOT **Lottery**: Lottery scheduling with a static allocation of tickets based on the maximum height of an item's utility function (UF). Randomly draw a ticket to decide what to show next.
- LOT-UF **Lottery with Utility Functions**: Lottery scheduling with dynamic allocation of tickets based on the current height of the item's UF. Randomly draw a ticket to decide what to show next.
- RAND **Random**: Out of all content items that would acquire some utility if shown next, randomly select which item to show.
- RR **Round Robin**: Show all content items in a circular order, skipping a content item if it would not acquire any utility.

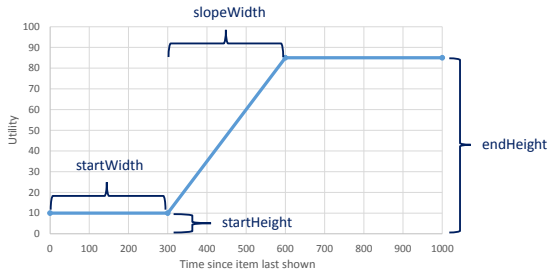
5.3 Workload Generation

We generated workloads for our evaluation using template utility functions. These template functions were designed to be simple functions with tunable parameters that allow for the generation of different workloads. The following experiments use these template utility functions, however, the lookahead algorithm does not depend on these templates. In practice, the UF for a content item can be any shape that adheres to the constraints listed in section 3.

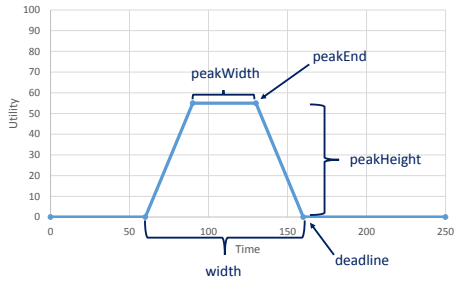
The template function for an AC item is defined by the four parameters shown in Figure 1a. It should be noted that the x-axis tracks the time passed since the content item was last shown. The intuition behind this utility function is that a content item will have a value of $startHeight$ immediately after being shown. This value will be close to zero because seeing the same content item twice within a short period of time is not useful to viewers. When the content item has not been shown for $startWidth$ seconds, the value of showing the content item begins to increase. After another $slopeWidth$ seconds, the utility function reaches its maximum possible value: $endHeight$. This value is indicative of the content item's overall usefulness to viewers. At the start of the simulation, the utility function of each AC item is seeded with a random time since last shown between 0 and 600 seconds.

The template function for a DC item is defined by the five parameters shown in Figure 1b. The intuition behind this utility function is that a DC item has a $deadline$ after which, showing the item is no longer useful to viewers. The content item is useful to viewers up to $width$ seconds before the deadline. However, showing this content item would be most valuable to viewers for a period of $peakWidth$ seconds ending at $peakEnd$. The value of the utility function for this period of maximum value is $peakHeight$. After this period of maximum value, the value of the utility function begins to decrease back to zero.

[3] https://www.wikipedia.org/wiki/Least_slack_time_scheduling



(a) AC template utility function



(b) DC template utility function

Figure 1: Template utility functions and parameters that enable content generation for different workloads

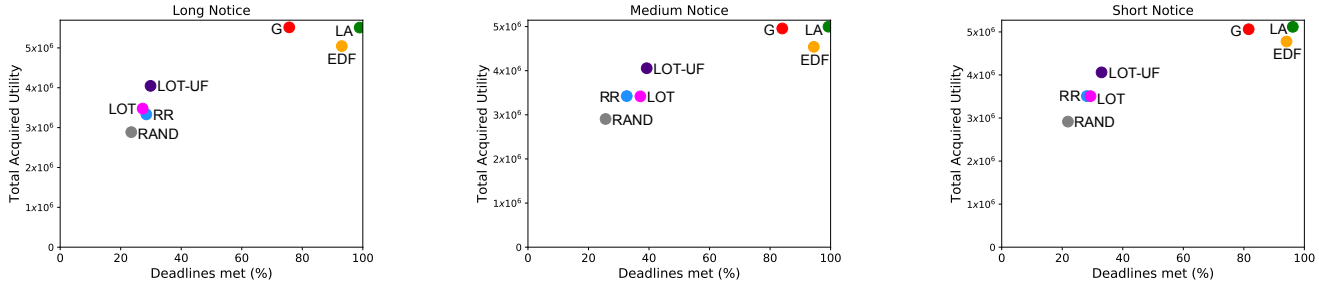


Figure 2: Total acquired utility and percent of DC deadlines met for running scheduling algorithms on workloads with different notice times. Long notice (left), medium notice (middle), and short notice (right). The notice time refers to how far in advance the content item is added to the scheduler library.

We generated a baseline workload that is realistic for the motivating example. Our workload consists of 15 AC items and 288 DC items with deadlines over the course of a 24 hour period (approx. 1 DC item added to the scheduler library every 5 min). The time when a content item is added to the scheduler library is its *awareTime*. At this time the content item will start being considered by the scheduler in its decisions. In the baseline workload, all AC items are included in the scheduler library from the beginning of the simulation. DC items are added to the scheduler library at some time before their deadlines. The parameters used for the utility functions in the baseline workload were randomly generated within the ranges shown in Table 1. In our experiments, we change certain parameters of the baseline workload to evaluate the performance of the lookahead algorithm across a variety of workloads.

5.4 Evaluation Metrics

To evaluate the performance of the scheduling algorithms, we consider two metrics: (a) *total acquired utility* and (b) *percentage of deadlines met*. Total acquired utility is the sum of the utility acquired by each content item that is shown over the course of the 24 hour period. The percentage of deadlines met is the percent of DC items that are shown and complete their full duration before their deadline. An algorithm that effectively integrates live content into the schedule would have a high total acquired utility and meet close to 100% of the deadlines. For the lookahead algorithm, we also evaluate the execution time, which is the average amount of time it takes to make a decision for which content item to show.

Table 1: Range of parameter values for utility functions of the baseline workload. For each content item, the value of the parameters of the UF are chosen from a uniform distribution that spans the range listed in the table. Times and durations are shown in seconds.

UF parameter		Baseline Range
DC	**duration**	(5, 60)
	deadline (t_d)	(1, 86400)
	width	($duration$, $duration * 8$)
	awareTime	($t_d - width - 300$, $t_d - width$)
	peakWidth	(0, $width$)
	peakEnd	($t_d - width + peakWidth$, t_d)
	peakHeight	(70, 100)
AC	**duration**	(5, 60)
	awareTime	0
	startWidth	(0, 600)
	slopeWidth	(0, 600)
	endHeight	(40, 80)
	startHeight	(0, $endHeight$)

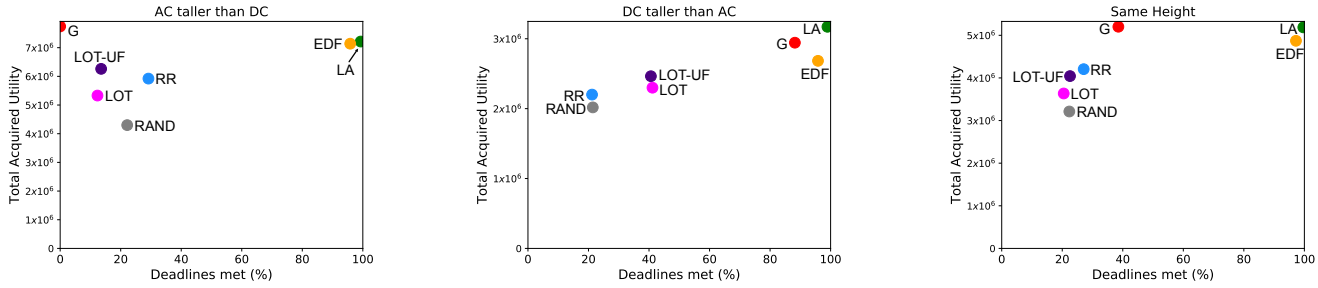


Figure 3: Total acquired utility and percent of DC deadlines met for running scheduling algorithms on workloads with different height UF's. AC taller than DC (left), DC taller than AC (middle), and AC same height as DC (right). The max height of a utility function is indicative of the overall importance of the content item to the viewer.

6 EXPERIMENTAL RESULTS

6.1 Notice time (Figure 2)

Notice time refers to the amount of time before the beginning of the utility function that a DC item is added to the scheduler library. When a content item has a longer notice time, the scheduler has more opportunity to schedule other content around it in a manner that maximizes the total utility. We evaluated the LA algorithm using workloads with long, medium, and short notice times. From the baseline workload, the range of DC *awareTime* was changed to ($deadline - width - 300$, $deadline - width - 240$), ($deadline - width - 180$, $deadline - width - 120$), and ($deadline - width - 60$, $deadline - width$) respectively. For all three workloads, the LA algorithm outperforms the other algorithms. Even with short notice times, the LA algorithm is able to meet 96% of deadlines and acquire more utility than all of the other algorithms evaluated.

6.2 Heights of AC vs DC (Figure 3)

The maximum height of the UF is an indicator of the general importance of the content item. We evaluated the LA algorithm using workloads where the heights of DC utility functions were taller, shorter, and the same height as the AC utility functions. From the baseline workload, the range of *peakHeight* for DC items was changed to (80, 100), (20, 40), and (50, 70) respectively. The range of *endHeight* for AC functions was changed to (20, 40), (80, 100), and (50, 70) respectively. When DC functions are taller than AC functions, the LA algorithm outperforms the other algorithms in terms of both acquired utility and deadlines met. When AC functions are taller than DC functions, the greedy algorithm acquires 7% more utility than the LA algorithm. However, it does so by not scheduling any DC items, thus meeting 99% fewer deadlines than the LA algorithm. Because the LA algorithm acquires very high utility and also integrates almost all of the live content into the schedule, it is the best performing algorithm for this workload too.

6.3 Number of DC items (Figures 4, 5)

The number of DC items is an indication of the scheduling difficulty. The more DC items there are, the more likely it is that there are overlapping UF's. When utility functions overlap, it is more difficult to create a schedule such that all items meet their deadlines and acquire high utility. We evaluated the LA algorithm using workloads with low, medium, high, and very high numbers of DC items. From the baseline workload, the number of DC items was changed to 288, 576, 864, and 1440 respectively. For all four workloads, the LA algorithm is able to acquire high utility while meeting deadlines.

6.4 Number of AC items (Figure 6)

We evaluated the LA algorithm using workloads with low (15), medium (30), and high (45) numbers of AC items. While the greedy, random, round robin, and lottery algorithms struggle to meet deadlines as the number of AC items increase, the LA algorithm is able to meet over 99% of the deadlines with all three workloads.

6.5 Lookahead window size (Table 2)

We evaluated the LA algorithm on the baseline workload using different lookahead window sizes. The optimal window size is dependent on the workload. The lookahead window should be at least as long as the longest duration content item. However, a slightly larger window allows the algorithm to consider more information when making a decision. The execution time of the algorithm increases linearly with the size of the window. The utility acquired by the algorithm increases logarithmically with the size of the window.

Table 2: Performance of the lookahead algorithm with different window sizes.

Window Size (sec)	Execution Time (ms)	% Deadlines Met	Total Acquired Utility
60	0.591	99.31%	4,434,015
90	0.817	98.61%	4,534,079
120	1.148	99.31%	4,614,187
150	1.634	98.96%	4,648,108
180	1.883	99.65%	4,651,114

7 DISCUSSION

Our experiments showed that the lookahead algorithm performs very well when time-based utility is the primary scheduling requirement. However, real-world public displays may have additional requirements that need to be considered. The filterer component of our architecture could be expanded to handle more requirements,

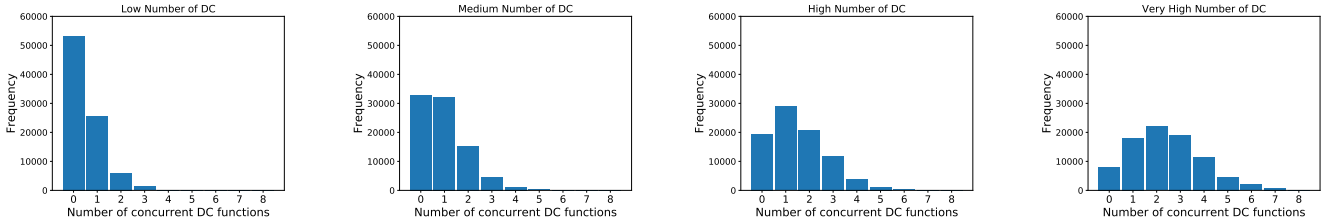


Figure 4: Distribution of the number of overlapping DC items over the 24 hour period (i.e. 2 means at a given second, there are two DC functions that have non-zero value). Distributions are shown for workloads with a low number of DC content (left), medium number (center left), high number (center right), and very high number (right).

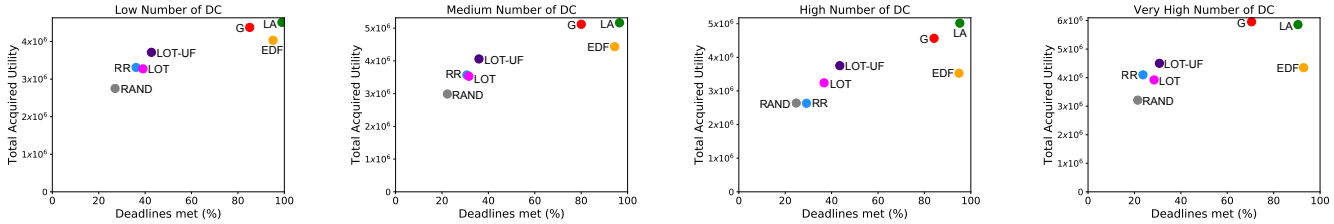


Figure 5: Utility acquired and deadlines met for running algorithms on workloads with the overlap distributions shown in Figure 4. Workloads with high numbers of DC items are more difficult to schedule because more utility functions overlap, making it more difficult to find an ordering of content that is optimal for all content items.

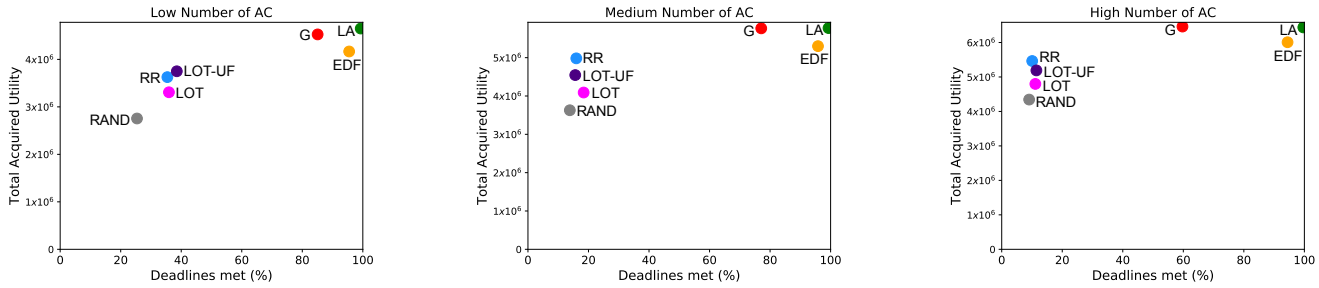


Figure 6: Total acquired utility and percent of DC deadlines met for running scheduling algorithms on workloads with different numbers of AC items. Small number (left), medium number (middle), and high number (right).

such as "Do not show B immediately after A". However, there are other requirements that our framework is currently unable to handle, such as "Show B immediately after A" or "Show A five times per day". Given prior usage studies, it is not clear that such features would be really useful to end-users [2].

One of the key challenges in using our proposed framework is coming up with utility functions for each content item. Rather than drawing or parameterizing the utility function, it may be more user-friendly to have a list of predetermined UF's to select from. It is ultimately up to display owners to design a utility function that they believe is representative of the value that the content item provides, but providing some simple "defaults" will go a long way in terms of usability.

8 CONCLUSIONS

Live content on public displays can provide immense value to passerbys. However, scheduling such content (together with static content) creates a number of challenges and, to the best of our knowledge, is not possible with current scheduling frameworks. In this work, we proposed the utility function framework and a novel lookahead algorithm that provide a mechanism for scheduling both live and static content items onto public displays. Our experiments showed that the algorithm is very effective in integrating live content (i.e., exhibiting very low rates of missed deadlines), while providing high utility to viewers.

ACKNOWLEDGMENTS

This work is part of the PittSmartLiving project which is supported by NSF award CNS-1739413.

REFERENCES

[1] Robert K. Abbott and Hector Garcia-Molina. 1992. Scheduling Real-time Transactions: A Performance Evaluation. *ACM Trans. Database Syst.* 17, 3 (Sept. 1992), 513–560. https://doi.org/10.1145/132271.132276

[2] Sarah Clinch, Nigel Davies, Adrian Friday, and Christos Efstratiou. 2011. Reflections on the Long-term Use of an Experimental Digital Signage System. In *Proceedings of the 13th International Conference on Ubiquitous Computing (UbiComp '11)*. ACM, New York, NY, USA, 133–142. https://doi.org/10.1145/2030112.2030132

[3] Nigel Davies, Sarah Clinch, and Florian Alt. 2014. Pervasive Displays: Understanding the Future of Digital Signage. *Synthesis Lectures on Mobile and Pervasive Computing* 8 (04 2014), 1–128. https://doi.org/10.2200/S00558ED1V01Y201312MPC011

[4] I. Elhart, M. Langheinrich, N. Davies, and R. José. 2013. Key challenges in application and content scheduling for Open Pervasive Display Networks. In *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. 393–396. https://doi.org/10.1109/PerComW.2013.6529524

[5] Ivan Elhart, Marc Langheinrich, Nemanja Memarovic, and Tommi Heikkinen. 2014. Scheduling Interactive and Concurrently Running Applications in Pervasive Display Networks. In *Proceedings of The International Symposium on Pervasive Displays (PerDis '14)*. ACM, New York, NY, USA, Article 104, 6 pages. https://doi.org/10.1145/2611009.2611039

[6] Jayant R. Haritsa, Miron Livny, and Michael J. Carey. 1991. Earliest Deadline Scheduling for Real-Time Database Systems. In *Proceedings of the Real-Time Systems Symposium - 1991, San Antonio, Texas, USA, December 1991*. IEEE Computer Society, 232–242. https://doi.org/10.1109/REAL.1991.160378

[7] E. Douglas Jensen, C. Douglas Locke, and Hideyuki Tokuda. 1985. A Time-Driven Scheduling Model for Real-Time Operating Systems. In *RTSS*.

[8] Kyoung-Don Kang, Sang Hyuk Son, and John A. Stankovic. 2004. Managing Deadline Miss Ratio and Sensor Data Freshness in Real-Time Databases. *IEEE Trans. Knowl. Data Eng.* 16, 10 (2004), 1200–1216. https://doi.org/10.1109/TKDE.2004.61

[9] Alexandros Labrinidis, Huiming Qu, and Jie Xu. 2007. Quality Contracts for Real-Time Enterprises. In *Lecture Notes in Computer Science 4365: Post Proceedings of First International Workshop on Business Intelligence for the Real Time Enterprise*. pp. 143–156. BIRTE'06 was held in conjunction with the VLDB'06 Conference, Seoul, Korea, Sept. 2006.

[10] Cynthia B. Lee and Allan E. Snavely. 2007. Precise and Realistic Utility Functions for User-centric Performance Analysis of Schedulers. In *Proceedings of the 16th International Symposium on High Performance Distributed Computing (HPDC '07)*. ACM, New York, NY, USA, 107–116. https://doi.org/10.1145/1272366.1272381

[11] Mateusz Mikusz, Sarah Clinch, and Nigel Davies. 2015. Are You Feeling Lucky?: Lottery-based Scheduling for Public Displays. In *Proceedings of the 4th International Symposium on Pervasive Displays (PerDis '15)*. ACM, New York, NY, USA, 123–129. https://doi.org/10.1145/2757710.2757721

[12] Jörg Müller, Juliane Exeler, Markus Buzeck, and Antonio Krüger. 2009. ReflectiveSigns: Digital Signs That Adapt to Audience Attention. In *Proceedings of the 7th International Conference on Pervasive Computing (Pervasive '09)*. Springer-Verlag, Berlin, Heidelberg, 17–24. https://doi.org/10.1007/978-3-642-01516-8_3

[13] Fernando Ribeiro and Rui Jose. 2010. Autonomous and Context-Aware Scheduling for Public Displays Using Place-Based Tag Clouds, Vol. 72. 131–138. https://doi.org/10.1007/978-3-642-13268-1_16

[14] Fernando Reinaldo Silva Garcia Ribeiro and Rui José. 2009. Timeliness for Dynamic Source Selection in Situated Public Displays. In *WEBIST*.

[15] Mohamed A. Sharaf, Shenoda Guirguis, Alexandros Labrinidis, Kirk Pruhs, and Panos K. Chrysanthis. 2008. ASETS: A Self-Managing Transaction Scheduler. In *Proc. of 3rd International Workshop on Self-Managing Database Systems*. pp. 56–62. held in conjunction with the 24th International Conference on Data Engineering (ICDE 2008), DOI:10.1109/ICDEW.2008.4498285.

[16] Oliver Storz, Adrian Friday, and Nigel Davies. 2006. Supporting content scheduling on situated public displays. *Computers and Graphics* 30, 5 (2006), 681–691. https://doi.org/10.1016/j.cag.2006.07.002

[17] Yukinobu Taniguchi. 2018. [Invited Paper] Content Scheduling and Adaptation for Networked and Context-Aware Digital Signage: A Literature Survey. *ITE Transactions on Media Technology and Applications* 6, 1 (2018), 18–29. https://doi.org/10.3169/mta.6.18

[18] Yukinobu Taniguchi, Hiroyuki Arai, Ken Tsutsuguchi, and Akihito Akutsu. 2014. Content-Schedule Optimization of Digital Signage Taking Account of Location Characteristics.