Utility Driven Optimization of Real Time Data Broadcast Schedules

Rinku Dewri[a,*], Indrakshi Ray[b], Indrajit Ray[b], Darrell Whitley[b]

[a] *Department of Computer Science, University of Denver, Denver, CO 80208, USA*
[b] *Department of Computer Science, Colorado State University, Fort Collins, CO 80523, USA*

Abstract

Data dissemination in wireless environments is often accomplished by on-demand broadcasting. The time critical nature of the data requests plays an important role in scheduling these broadcasts. Most research in on-demand broadcast scheduling has focused on the timely servicing of requests so as to minimize the number of missed deadlines. However, there exists many environments where the utility of the received data is an equally important criterion as its timeliness. Missing the deadline may reduce the utility of the data but does not necessarily make it zero. In this work, we address the problem of scheduling real time data broadcasts with such soft deadlines. We investigate search based optimization techniques to develop broadcast schedulers that make explicit attempts to maximize the utility of data requests as well as service as many requests as possible within an acceptable time limit. Our analysis shows that heuristic driven methods for such problems can be improved by hybridizing them with local search algorithms. We further investigate the option of employing a dynamic optimization technique to facilitate utility gain, thereby eliminating the requirement of a heuristic in the process. An evolution strategy based stochastic hill-climber is investigated in this context.

Keywords: broadcast scheduling, data utility, local search, evolution strategy, quality of service

1. Introduction

Wireless computing involves a network of portable computing devices thoroughly embedded in our day-to-day work and personal life. The devices interact with each other and with other computing nodes by exchanging rapid and continuous streams of data. To facilitate almost imperceptible human-computer interaction, data access times in such environments must be maintained within a specified quality-of-service (QoS) level. Challenges in doing so arise from the fact that wireless bandwidth is typically a limited resource, and thus it is not always possible to meet the quality requirements of every device. This constraint not only makes wireless data access a challenging problem, but also identifies "optimal resource allocation" as one of the major research problems in this domain.

A wireless environment encompasses both peer-to-peer and client-server modes of data dissemination. For example, a typical pervasive health care system may involve multiple sensor nodes disseminating data on the monitored vital signs of a patient to a personal digital assistant carried by the attending health care personnel. Data communication follows a peer-to-peer architecture in such a setting. On the other hand, a wireless environment designed to serve queries on flight information in an airport is based on a client-server mode of communication. Flight information usually reside in a database server from where data is disseminated based on the incoming queries. For an environment like an airport, it is appropriate to assume that the database will be queried more frequently for certain types of data. Similar situations can be imagined in a stock trading center, a pervasive traffic management system, or a pervasive supermarket. Such scenarios open up possibilities of adopting a broadcast based architecture to distribute data in a way that multiple queries for the same data item get served by a single broadcast. The focus of this

*Corresponding author: Tel. +1-303-871-4189; Fax. +1-303-871-3010

Email addresses: rdewri@cs.du.edu (Rinku Dewri), iray@cs.colostate.edu (Indrakshi Ray), indrajit@cs.colostate.edu (Indrajit Ray), whitley@cs.colostate.edu (Darrell Whitley)

paper is directed towards a combinatorial problem that arises in this approach.

1.1. An Example

Quality of service (QoS) is an important facet in wireless data access. Consider the following example application - a traffic management system for a big city. The city government implements a traffic pricing policy for all vehicles on all roads based on factors such as the distance traveled, the type of road traveled on, the time of day, vehicle category and customer type (for example, special fee paying, traveling salesman, paramedic on-call, etc.) The system gathers real time traffic data via thousands of sensors, such as traffic cameras, magneto-resistive traffic sensors and radars spread throughout the city. To help drivers optimize their travel costs, the traffic management system provides various routing services to drivers to avoid roadblocks, construction delays, congestion, accidents etc[1].

A driver requests and gets routing services using smart GPS equipped devices as follows. A device periodically uploads the vehicle's current location (based on GPS information), intended destination, time willing to spend for traveling to destination, a prioritized list of routing restrictions (for example, waypoints that the driver would like to visit if possible, rest stops, scenic byways, etc.), vehicle category and customer type, and current speed. In response, the server replies with a set of route information. Each route information contains, among other things, information about road segments to travel on this route and traffic patterns on these road segments. The GPS equipped device uses this information and uses other parameters set by the driver to compute a good route to take. Note that since the response to many drivers will no doubt contain overlapping route segments and traffic information, it makes sense broadcasting this data.

The requests arrive at the server with various priority levels and soft deadlines. For example, a higher fee paying customer gets a higher priority than a lesser fee paying customer. A VIP's convoy may get a still higher priority. Emergency responders get the highest priority. A routing request that comes associated with a set of routing restrictions, for example, a list of waypoints that the driver would like to visit en route, automatically gets associated with a set of soft deadlines based on the speed of the driver. The requests from different drivers may also get re-prioritized at the server so as to meet a broader goal of reducing congestion and enabling smoother traffic flows.

In this example, the different data that the server needs to serve is associated with different utility values. Owing to the dynamic nature of the utility of responses to queries, the time criticality factor cannot be ignored altogether when disseminating data. The server would like to satisfy as many queries in a timely manner as possible. However, some queries may be delayed beyond their specified deadlines (for example, the query from the VIP's convoy). The users, who hardly realize the broader goals of the traffic management system and various bottlenecks in the information infrastructure, would like to have their requests served at the earliest; however, it is reasonable to assume that delayed data still provide some utility even if received after a specified deadline. For example, a delayed route information may prevent a driver from visiting a particular waypoint en route or may require the driver to use the next gas station. Nonetheless, it may still allow the driver to choose a good route to the destination. An assumption of reduced data utility in the face of missed deadline enables data broadcasts to be tailored in such a way that total utility associated with a broadcast is maximized. This helps maintain a certain QoS level in the underlying infrastructure.

Note that the specific problem we are trying to address is by no means restricted only to the example application outlined above. Similar scenarios are witnessed in environments such as stock exchanges. Here stock brokers on the floor seek and receive periodic market data on wireless hand-held devices and notebooks and analyze them to determine which stocks are rewarding. They also buy and sell stocks using such devices. Popularity of stocks changes throughout the day, and it is important to analyze such trends along multiple dimensions in order to buy and sell stocks. Thus, although queries from brokers are implicitly associated with deadlines, these can be considered soft. The overall goal of the stock exchange still remains to satisfy as many requests as possible in a timely manner. To wrap up the scope of the problem domain we would like to point out that in order to make useful decisions, the stock broker may make a request for a group of data from the stock exchange (for example, stock prices of three different

[1]Note that such an application is not very far-fetched. The Dutch government is in the process of introducing a similar electronic traffic management system and pricing model. See [1].

oil companies). A typical constraint on such a request is that all these data items must be received (not necessarily in any particular order) before the stock broker can perform the relevant local analysis of the market trends. Such a request can be considered as a *transactional request* (or simply a *transaction*). For scheduling in such cases, additional constraints must be placed for ensuring the validity of data received.

1.2. Contributions

Wireless broadcast mechanisms have been extensively investigated earlier. However, very few of the proposed approaches give attention to the effective utility involved in the timely servicing of a request. Time criticality has been earlier addressed in a number of contexts with the assumption of a *hard deadline* [2, 3, 4]. Broadcast scheduling in these works mostly focus on the timely servicing of a request to minimize the number of missed deadlines. When the assumption of a *soft deadline* is appropriate, a broadcast schedule should not only try to serve as many requests as possible, but also make a "best effort" in serving them with higher utility values. Often, heuristics are employed in a dynamic setting to determine these schedules. However, their designs do not involve the QoS criteria explicitly. Heuristic based methods make local decisions w.r.t. a request or a broadcast, and often fail to capture the sought global QoS requirement. Much of this is due to the fact that real time decision making cannot span beyond an acceptable time limit, thereby restricting the usage of "full length" optimization techniques to the domain. It does become imperative to design hybrid strategies that can combine the fast real time performance of heuristics and the better solution qualities obtained from search based optimization.

Our contributions in this paper are summarized in the following points.

1. We explore data broadcasting as a combinatorial problem defined with an objective of utility maximization. This utility driven optimization of broadcasts adds more value to a wireless system since the energy cycles used by the devices while retrieving a data item will presumably not be wasteful.
2. We provide an extensive analysis of the search space involved in the problem and report the performance of multiple heuristics on a set of statistically generated synthetic requests. We argue that traditional heuristics usually generate solutions in a sub-optimal part of this space with respect to a given global utility measurement.
3. We empirically demonstrate that "local search" can be used in real-time to boost the performance of naive heuristics such as EDF, with performance levels often at par with other elaborate heuristics. In addition, we propose an evolution strategy based light-weight stochastic hill-climber that explicitly searches the space of schedules to maximize utility. Results demonstrate that, in certain problem types, this explicit search strategy can generate higher objective values than a traditional heuristic based approach.
4. We perform the analysis indicated in the above points in the context of two problem types, namely at the data item level and at the transaction level. In the former type, requests involve a single data item, while multiple unordered data items are involved in a request of the latter type. Besides the local search and evolution strategy based approaches, we present the performance of heuristics such as EDF, HUF, RxW, SIN-α, NPRDS and ASETS* on one or more of these problem types. Transaction level scheduling induces an exponentially large search space and has been observed to demonstrate very different dynamics than a data item level problem.

The rest of the paper is organized as follows. Section 2 summarizes the related work in this domain. The broadcast model and the scheduling problem are discussed in Section 3. The explored solution methods are described in Section 4. Section 5 discusses the scheduling problem at the transaction level. The experimental setup followed by results and discussions are summarized in Section 6 and Section 7, respectively. Finally, Section 8 concludes the paper.

2. Related Work

Data broadcasting has been extensively studied in the context of wireless communication systems. Su and Tassiulas [5] study the problem in the context of access latency and formulate a deterministic dynamic optimization problem, the solution to which provides a minimum access latency schedule. Acharya and Muthukrishnan propose the *stretch*

metric [6] to account for differences in service times arising in the case of variable sized data items. They propose the MAX heuristic to optimize the worst case stretch of individual requests. Aksoy and Franklin's RxW heuristic [7] is an attempt to combine the benefits of the MRF and FCFS heuristics, each known to give preference to popular and long standing data items, respectively. Hameed and Vaidya adapt a *packet fair queuing* algorithm to this domain [8]. Lee et al. provide a survey of these techniques [9] and their applicability in the area of pervasive data access.

The above mentioned algorithms ignore the time criticality factor while serving data requests. Early work on time constrained data requests is presented by Xuan et al. [10]. Jiang and Vaidya address the issue by considering broadcast environments where clients do not wait indefinitely to get their requests served [2]. Lam et al. look at the time criticality factor from the perspective of temporal data validity [11]. Fernandez and Ramamritham propose a hybrid broadcasting approach to minimize the overall number of deadlines missed [12]. Temporal constraints on data are revisited by Wu and Lee with the added complexity of request deadlines [13]. Their $RDDS$ algorithm assigns a priority level to each requested data item and broadcasts the item with the highest priority. Xu et al. propose the SIN-α algorithm to minimize the request drop rate [4]. However, their approach does not take variable data sizes into consideration. This motivates Lee et al. to propose the $PRDS$ algorithm that takes into account the urgency, data size and access frequencies of various data items [3].

Theoretical analysis on the hardness of the problem has led to a number of conclusions in the recent few years. In particular, Chang et al. show that the offline problem of minimizing the maximum response time is NP-complete [14]. In the real-time case, the authors show that no deterministic algorithm can be better than 2-competitive. Similarly, broadcast scheduling with the objective of minimizing the total response time, or with deadlines, has also been proved to be NP-complete.

Several shortcomings of using a strict deadline based system are discussed by Ravindran et al. [15] in the context of real-time scheduling and resource management. Jensen et al. point out that real-time systems are usually associated with a value based model which can be expressed as a function of time [16]. They introduce the idea of *time-utility functions* to capture the semantics of soft time constraints that are particularly useful in specifying utility as a function of completion time. An attempt to understand the benefit of utility values in hard deadline scheduling algorithms is undertaken by Buttazzo et al. [17]. Wu et al. study a task scheduling problem where utility is considered a function of the start time of the task [18]. Utility based scheduling in data broadcasting is addressed by Dewri et al. in the context of pervasive environments [19]. However, the focus of the work is limited to requests for single data items only. The work presented here provides extensions in the form of formulations catering to the scheduling problem when requests are made at the transaction level.

While the notion of a transaction have been explored in the database community, its treatment in the form of a collection of data items is rare in the broadcasting domain. Chehadeh et al. take into consideration object graphs to cluster related objects close to one another in the broadcast channel [20]. Hurson et al. propose an extension for multiple broadcast channels [21]. Huang et al. show that several cases of the problem of broadcasting related data is NP-hard and propose a genetic algorithm to address the problem [22]. Guirguis et al. propose the ASETS* algorithm to schedule workflows that is conceptually equivalent to scheduling multiple data items with an ordering constraint [23]. This work caters to a different category of applications where such precedence constraints are not always available.

Multiple issues related to data broadcasting, such as hard deadlines, request drop rate, data item size, and access frequencies, have been addressed in this collection of works. We also consider variable data sizes and their access frequencies in the simulation study, but do not measure the performance of an algorithm using the request drop rate. Instead, a time-utility function is used for this purpose.

3. Broadcast Scheduling

Wireless data broadcasting is an efficient approach to address data requests, particularly when similar requests are received from a large user community. Broadcasting in such cases alleviate the requirement for repeated communication between the server and the different clients interested in the same data item. *Push-based* architectures broadcast commonly accessed data at regular intervals. Contrary to this, *on-demand* architectures allow

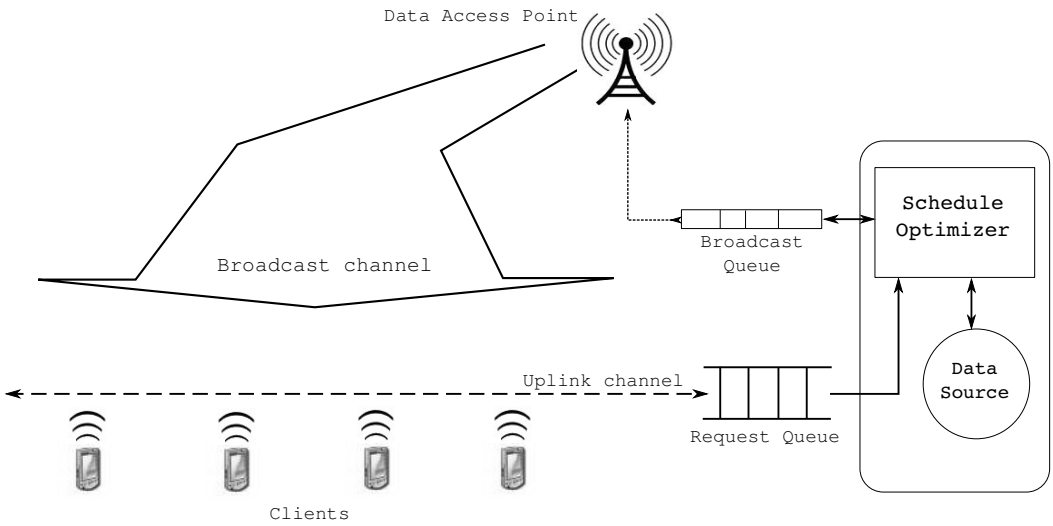


Figure 1: Typical on-demand broadcast architecture in a wireless environment.

the clients to send their requests to the server. However, access to the data item is facilitated through a broadcast which, for more frequently requested data, serves multiple clients at a time.

Data access in wireless environments can be modeled with an on-demand broadcast architecture where particular emphasis has to be paid to the time criticality and utility of a served request. The time criticality factor emphasizes that the requested data is expected within a specified time window; failure to do so would result in an utility loss. A broadcast schedule in such environments has to cater to the added requirement of maintaining a high utility value for a majority of the requests.

3.1. Broadcast Model

Fig. 1 shows a typical on-demand data broadcast architecture in a wireless environment. Different client devices use an uplink channel to a data provider to request various data items served by the provider. The data items are assumed to reside locally with the data provider. Each request Q_j takes the form of a tuple $\langle D_j, R_j, P_j \rangle$, where R_j is the response time within which the requesting client expects the data item D_j and asserts a priority level P_j on the request. Although expected response times and priorities can be explicitly specified by the client, it is not a strict restriction. For example, the broadcast server can assign priorities and response time requirements to the scheduler based on the current status (say a geographic location) of the client making the request. Note that a single request involves only one data item. A client requiring multiple data items sends multiple requests for each data item separately. Requests from the same client can be served in any order. The data provider reads the requests from a queue and invokes a scheduler to determine the order in which the requests are to be served. It is important to note that new requests arrive frequently into the queue which makes the task of the scheduler rather dynamic in nature. The scheduler needs to re-examine the previously generated schedule to accommodate the time critical requirements of any new request. Data dissemination is carried out through a single channel data access point. Clients listen to this channel and consider a request to be served as soon as the broadcast for the corresponding item begins.

The underlying scheduler is invoked every time a new request is received. At each invocation, the scheduler first determines the requests that are being currently served and removes them from the request queue. The data items required to serve the remaining requests are then determined and a schedule is generated to serve them. The scheduler tries to make a "best effort" at generating a schedule that respects the response time requirements of the requesting devices.

3.2. Utility Metric

Real-time systems are typically known to utilize the abstraction of hard deadlines to specify time constraints during resource management. As pointed out in [15], such abstractions result in vari-

ous drawbacks while making a scheduling decision. The most prominent of all is the *domino* effect during overload conditions [24] where a hard deadline based scheduler would start favoring jobs that are near to missing their deadlines. Depending on the completion time of such favored jobs, a situation can arise where the scheduler is no longer (or for a majority) able to schedule jobs that satisfy their deadlines. Further, hard deadlines are usually binary in nature – they are either met or not met. This prohibits any distinction to be made on the quality of schedules that differ primarily on the amount of delay introduced in meeting the time constraints. Note that such distinctions are important for a scheduler incorporating some form of search to find better schedules. Hard deadlines also fail to capture any non-linear time constraint where the utility of completing a job is related to the time of completion, as has been pointed out earlier in the motivating example.

Utility functions overcome these problems by adapting their behavior in times of overload scenarios. In underload conditions, utility functions drive schedules towards urgency satisfaction (meeting deadlines) as the highest utility is typically associated with the time instances within the deadline. When an overload occurs (deadlines are difficult to meet), a summed utility value satisfying an optimality criteria offers a schedule that favors jobs that are more important to be completed over those which are more urgent, in that they can generate higher utility. In fact, strict deadline based scheduling incorporates a very restricted formulation of an utility function.

The utility of a data item broadcast is measured from the response time and priority level of the requests served by it. The response time r_j of a request Q_j arriving at time $t_{j,arr}$ and served at time $t_{j,ser}$ is given by $(t_{j,ser} - t_{j,arr})$. As in [16], we assume that the utility of a data item received by a client decreases exponentially if not received within the expected response time (Fig. 2). For a given request Q_j, the utility generated by serving it within a response time r_j is given as,

$$u_j = \begin{cases} P_j & , r_j \leq R_j \\ P_j e^{-\alpha_j (r_j - R_j)} & , r_j > R_j \end{cases} \quad (1)$$

The utility of broadcasting a data item d is then given as,

$$U_d = \sum_{j | d \ serves \ Q_j} u_j \quad (2)$$

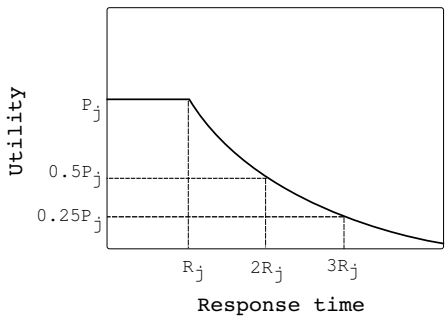


Figure 2: Utility of serving a request.

For a given schedule S that broadcasts the data items $D_1, D_2, \ldots, D_k$ in order, the utility of the schedule is given as,

$$U_S = \sum_{d=D_1}^{D_k} U_d \quad (3)$$

In this work, we assume that the utility of a data item for a client decays by half for every factor of increase in response time, i.e. $\alpha_j = \ln 0.5 / R_j$.

3.3. Problem Statement

A data source D is a set of N data items, $\{D_1, D_2, \ldots, D_N\}$, with respective sizes $d_1, d_2, \ldots, d_N$. A request queue at any instance is a dynamic queue Q with entries Q_j of the form $\langle D_j, R_j, P_j \rangle$, where $D_j \in D$, and $R_j, P_j \geq 0$. At an instance t_{curr}, let $Q_1, Q_2, \ldots, Q_M$ be the entries in Q. A schedule S at an instance is a total ordering of the elements of the set $\bigcup_{j=1,\ldots,M} \{D_j\}$.

In the context of the broadcast scheduling problem, the request queue Q corresponds to the queue left after all requests currently being served are removed. Note that two different entries Q_j and Q_k in Q may be interested in the same data item. However, it is important to realize that the current instance of the scheduler only needs to schedule a single broadcast for the data item. The arrival time of all requests in Q is at most equal to the current time, t_{curr}, and the scheduled broadcast time t for the data item in Q will be t_{curr} at the earliest. A schedule is thus a total ordering of the unique elements in all data items requested.

The time instance at which a particular data item from the schedule starts to be broadcast depends on the bandwidth of the broadcast channel. A broadcast channel of bandwidth b can transmit b *data units* per *time unit*. If t_{ready} is the ready time of the

channel (maximum of t_{curr} and the end time of current broadcast), then for the schedule $D_1 < D_2 < \ldots < D_k$, the data item D_i starts to be broadcast at time instance $t_{D_i} = t_{ready} + \sum_{j=1}^{i-1}(d_j/b)$. All requests in Q for the data item D_i is then assumed to be served, i.e. $t_{j,ser}$ for such requests is set to t_{D_i}. We explicitly mention this computation to point out that the utility metric involves the *access time*, and not the *tuning time*, of a request. The access time is the time elapsed from the moment a client issues a request to the moment when it starts to receive the requested data item. The tuning time is the time the client has to actively scan the broadcast channel to receive the data item.

While Eq. (3) can be used to measure the instantaneous utility of a schedule, it is not suitable in determining the performance level of a solution method in a dynamic setting. We thus use the utility generated from the queries already served as the yardstick to compare performance. In other words, the performance of a solution methodology at an instance where queries $Q_1, \ldots, Q_K$ are already served is measured by $\sum_{j=1}^{K} u_j$. In effect, we are interested in the global utility generated by the method. The aforementioned QoS criteria could be a specification in terms of this global utility. The objective behind the design of a solution methodology is to maximize this global utility measured at any instance.

4. Solution Methodology

An implicit constraint in real time data broadcasting is the time factor involved in making a scheduling decision. Data requests usually arrive more frequently than they can be served, which in turn leads to the generation of a long request queue. Any scheduling method must be fast enough to generate a good schedule without adding much to the access time of requests. Latencies induced between broadcasts because of the scheduling time is also a detrimental factor to resource utilization. Heuristics are often used as fast decision makers, but may result in degraded solution quality. Hybrid approaches in this context can provide a suitable trade-off between solution quality and decision time.

Operating systems employ a range of heuristics for various resource scheduling tasks. However, these scheduling tasks have some inherent differences with data broadcast scheduling. For instance, schedulers used for CPU scheduling assume that a scheduled entity will have exclusive access to the processor. This assumption is not valid in data broadcasting since a scheduled data item can very well serve multiple requests. Most schedulers also do not incorporate time-criticality constraints in their design. Algorithms such as C-SCAN ensures that all I/O requests are eventually served, but does not consider the priority of a request. The operating system has to be policy driven to enforce the priority requirements.

4.1. Heuristics

For the purpose of this study, we use two traditional heuristics – *Earliest Deadline First* (EDF) and *Highest Utility First* (HUF) – which takes into account the time critical nature of a data request. Further, we experiment with three state-of-the-art heuristics proposed for deadline based scheduling.

4.1.1. EDF and HUF

EDF starts by first scheduling the data item that corresponds to a request with the minimum $t_{curr} - (t_{arr} + R_j)$. All requests in Q that get served by this broadcast are removed (all requests for the scheduled data item) and the process is repeated on the remaining requests. For multiple channels, the heuristic can be combined with *best local earliness* to map the data item to a channel that becomes ready at the earliest. EDF gives preference to data items that have long been awaited by some request, thereby having some level of impact on the utility that can be generated by the requests. However, it does not take into account the actual utility generated. HUF alleviates this problem by first considering the data item that can generate the highest amount of utility. The strategy adopted by HUF may seem like a good approach particularly when the overall utility is the performance metric. However, HUF generated schedules may not be flexible enough in a dynamic environment. For example, if the most requested data item in the current request queue generates the highest utility, HUF would schedule the item as the next broadcast. If this data item requires a high broadcast time, not only will the subsequent broadcasts in the schedule suffer in utility, but new requests will also have to wait for a long time before getting served.

4.1.2. RxW

The RxW heuristic [7] schedules data items in decreasing order of their $R \times W$ values, where R is the number of pending requests for a data item and W

is the time for which the oldest pending request for the data item has been waiting. Such a broadcast mechanism gives preference to data items which are either frequently requested or has not been broadcast in a long time (with at least one request waiting for it). This approach aims at balancing popularity and accumulation of requests for unpopular items. Hence, although the heuristic does not have any implicit factor that considers the deadline of requests, deadlines can be met by not keeping a request too long in the request queue.

4.1.3. SIN-α

The SIN-α (Slack time Inverse Number of pending requests) heuristic [4] uses the following two arguments.

- *Given two items with the same number of pending requests, the one with a closer deadline should be broadcast first to reduce request drop rate;*
- *Given two items with the same deadline, the one with more pending requests should be broadcast first to reduce request drop rate.*

Based on these two arguments, the $sin.\alpha$ value for a data item (requested for by at least one client) is given as

$$sin.\alpha = \frac{slack}{num^{\alpha}} = \frac{1stDeadline - clock}{num^{\alpha}}$$

where *slack* represents the urgency factor, given as the duration from the current time (*clock*) to the absolute deadline of the most urgent outstanding request (*1stDeadline*) for the data item, and *num* (≥ 1) represents the productivity factor, given as the number of pending requests for the data item. With $sin.\alpha$ values assigned to the data items, the schedule is created in increasing order of the $sin.\alpha$ values. The α value in our experiments is set at 2 based on overall performance assessment presented in the original work. We shall henceforth refer to this heuristic as SIN2.

4.1.4. NPRDS

Apart from the two arguments provided for SIN-α, the PRDS (Preemptive Request count, Deadline, Size) heuristic [3] incorporates a third factor based on data sizes.

- *Given two data items with the same deadline and number of pending requests, the one with a smaller data size should be broadcast first.*

We modify PRDS into a non-preemptive version and call it NPRDS. The heuristic works by first assigning a priority value to each data item (with at least one request for it), given as $\frac{R}{dln \times s}$, where R is the number of pending requests for the data item, dln is the earliest feasible absolute deadline (the broadcast of the item can serve the request before its deadline) of outstanding requests for the data item, and s is the size of the item. The NPRDS schedule is generated in decreasing order of the priority values. The heuristic aims at providing a fair treatment to different data sizes, access frequency of data items and the deadline of requests.

4.2. Heuristics with Local Search

As mentioned earlier, one goal of this work is to understand the performance of heuristics when coupled with local search techniques. We therefore introduce some amount of local search to improve the schedules generated by the heuristics. The notion of local search in this context involves changing the generated schedules by a small amount and accept it as the new schedule if an improvement in the utility of the schedule is obtained. The process is repeated for a pre-specified number of iterations. Such a "hill-climbing" approach is expected to improve the utility of the schedule generated by a heuristic. We employ the 2-exchange operator to search a neighborhood of the current schedule. The operator randomly selects two data items and swaps their positions in the schedule (Fig. 3). The notations EDF/LS and HUF/LS denote EDF and HUF coupled with local search, respectively. Intuitively, these hybrid approaches should provide sufficient improvements over the heuristic schedules, w.r.t. the performance metric, as the local search would enable the evaluation of the overall schedule utility, often ignored when using the heuristics alone.

```
Old Schedule    : a b c d e f g h i j
Swap Pts        :   *     *
New Schedule    : a e c d b f g h i j
```

Figure 3: 2-exchange operator example.

4.3. $(2+1)$*-ES*

Evolution Strategies (ES) [25, 26] are a class of stochastic search methods based on computational models of adaptation and evolution. They are typically expressed by the μ and λ parameters signifying the parent and the child population, respectively. Whereas the algorithmic formulation of evolution strategies remains the same as that of a genetic algorithm [27], two basic forms have been defined for them. In the $(\mu + \lambda)$-ES, μ best of the combined parent and offspring generations are retained using truncation selection. In the (μ, λ)-ES variant, the μ best of the λ offspring replace the parents. These definitions are analogous to that of the steady-state and generational variants of genetic algorithms. The steady-state variant of genetic algorithms explicitly maintains the best solution found so far, while the generational variant blindly replaces the current population with the offspring population generated.

For our experimentation, we employ the $(\mu + \lambda)$-ES variant with $\mu = 2$ and $\lambda = 1$. This simple form of the ES is chosen to keep the dynamic scheduling time within an acceptable limit without sacrificing on the solution quality. Also, a $(2 + 1)$-ES can be seen as a type of greedy stochastic local search. It is stochastic because there is no fixed neighborhood and therefore the neighborhood does not define a fixed set of local optima. Otherwise, the method is very much a local search technique – sample the neighborhood and move to the best point. The pseudo code for the algorithm is given below.

Step 1: Generate two initial solutions x and y and evaluate them.

Step 2: Recombine x and y to generate an offspring.

Step 3: Mutate the offspring with probability p.

Step 4: Evaluate the offspring.

Step 5: Replace x and y by the two best solutions from x, y and the offspring.

Step 6: Goto Step 2 until termination criteria is met.

4.3.1. Solution encoding and evaluation

For the data broadcasting problem mentioned in the previous section, a typical schedule can be represented by a permutation of the unique data item numbers in Q. Thus, for n unique data items, the search spans over a space of $n!$ points. The evaluation of a solution involves finding the utility of the represented schedule as given by Eq. (3). The higher the utility, the better is the solution.

4.3.2. Syswerda's recombination operator

Recombination operators for permutation problems differ from usual crossover operators in their ability to maintain the uniqueness of entries in the offspring produced. For a schedule encoded as a permutation, it is desired that recombination of two schedules does not result in an invalid schedule. A number of permutation based operators have been proposed in this context [28]. We employ Syswerda's recombination operator in this study. The operator is particularly useful in contexts where maintaining the relative ordering between entries is more critical than their adjacency.

```
x         :  a b c d e f g h i j
y         :  c f a j h d i g b e
Key Pos.  :     * *     *   *
Offspring:  a j c d e f g h i b
```

Figure 4: Syswerda's recombination operator.

The operator randomly selects several key positions and the order of appearance of the elements in these positions are imposed from one solution to the other. In Fig. 4, the entries at the four key positions from y, $\{a, j, i, b\}$, are rearranged in x to match the order of occurrence in y. The offspring is x with the rearranged entries.

4.3.3. Mutation using insertion

An elementary mutation operator defines a certain neighborhood around a solution which in turn dictates the number of states which can be reached from the parent state in one step [25]. Insertion based mutation selects two random positions in a sequence and the element at the first chosen position is migrated to appear after the element in the second chosen position (Fig. 5).

```
Parent    : a b c d e f g h i j
Mutate Pts:   *     *
Offspring : a c d e b f g h i j
```

Figure 5: Mutation using the insertion operator.

4.3.4. Initialization and termination

The initial solutions determine the starting points in the permutation space where the search begins. Thus, a good solution produced by EDF or HUF could be a choice for the purpose. However, we do not want to add the complexity of determining an EDF or HUF schedule to the search process, and hence randomly generate the initial solutions x and y. The ES algorithm is terminated after a fixed number of iterations. If schedules generated by other heuristics are taken as initial solutions, the termination criteria could as well be specified as the point where a particular level of improvement has been obtained over the starting schedules. It is important that an alternative is also suggested since improvement based termination may never get realized.

5. Transaction Scheduling

The aforementioned problem assumes that requests are made at the data item level. A client interested in multiple data items would make independent requests for the individual items. As an extension of this problem, we next consider the case of *transaction level* requests. As mentioned in Section 1, a transaction level request differs from a data item level request in the sense that a request involves more than one data item. The order of retrieval of the data items is not important, but processing at the client end cannot start until all data items are received.

The formulation of this next problem is similar to the one suggested in Section 3.3, with differences appearing in the definition of a query Q_j. An entry Q_j in the request queue now takes the form $\langle \mathcal{D}_j, R_j, P_j \rangle$, where $\mathcal{D}_j = \{D_{j_1}, \ldots, D_{j_n}\} \subseteq D$ and $R_j, P_j \geq 0$. The request is considered to be served at $t_{j,served}$, which is the time instance when the last remaining data item in $\mathcal{D}_j$ is broadcast. The utility generated by serving the request then follows from Eq. (1). Note that, with this formulation, scheduling at the transaction level is not a simple extension of the problem of scheduling an aggregation of multiple requests from the same client. The following factors highlight the differences in data item and transaction level scheduling.

1. Multiple data item level requests made from the same client can have different deadlines associated with different data items. However, the deadline specified in a transaction level request is specific to the transaction and not the individual data items required to complete it.
2. Each request made at the data item level would accrue some utility, irrespective of whether it came from the same client or not. The notion of dependency between the data items (to complete a transaction) is not embedded in the requests. A request at the transaction level specifies this dependency by collecting the required data items into a set and making a request for the entire set. Utility is accrued in this case only when all items in the set are served.
3. When requests are made independently, the scheduler is not required to maintain low levels of latency between broadcasts of data items requested by the same client. However, at the transaction level, the schedules generated must take the latency into consideration since the utility will keep reducing until all requested data items are retrieved by the client.

5.1. Heuristics

The two traditional heuristics for data item level scheduling – EDF and HUF – are slightly modified to perform transaction level scheduling. The EDF heuristic for transaction level scheduling, denoted by T-EDF, sequentially schedules all data items in a request by giving preference to the request having the earliest deadline. The HUF heuristic for transaction level scheduling, denoted by T-HUF, sequentially schedules all data items in a request with preference to the one which can generate the highest amount of utility. The modifications enable the heuristics to make scheduling decisions based on the deadline, or utility, of serving a request, rather than the data items contained in it.

One might argue that T-EDF and T-HUF can be represented by EDF and HUF, respectively by considering each request to be for a single data item whose size is given by the total size of the data items requested. This is a viable representation since T-EDF and T-HUF do not take the data items contained in a request into account. However, note that the schedule generated is always at the data item level. Hence, there is room to exploit any existing overlaps in the data items of two different requests. For example, consider the requests A for data items $\{D_1, D_2, D_3, D_4\}$ and B for data items $\{D_2, D_4\}$, with every data item having the same size. Representing A as a request for

a data item D_A (whose size is the sum of the sizes of D_1, D_2, D_3 and D_4) and B as a request for a data item D_B loses the fact that the data items in B are a subset of those in A. Further, exploiting the existence of such subsets is also not trivial. This is specifically true for T-EDF which makes decisions based solely on the deadline. If A has an earlier deadline than B then the schedule generated will be $D_1 < D_2 < D_3 < D_4$; otherwise $D_2 < D_4 < D_1 < D_3$. Observe that, in the former case, both requests A and B will be served at the same time instance (the instance when D_4 starts broadcasting), while in the latter case, B will get served earlier. Hence, irrespective of the deadlines, the latter schedule is always equal to or better than the former. It is possible that even T-HUF fails to exploit the relationship. If serving A first generates higher utility than serving B first, T-HUF would generate the former schedule, thereby pushing the finish time of B further in the time line. It misses the observation that by serving B first it can still maintain the same utility for A. The situation is further complicated when the data items have varying sizes. Given such observations, augmenting the heuristics with local search would allow for an interesting exploration at the data item level. Recall that the local search would attempt to improve on the heuristic generated schedule by swapping data items at different positions. The corresponding local search variants for T-EDF and T-HUF are denoted by T-EDF/LS and T-HUF/LS, respectively.

5.2. $(2+1)$-*ES*

The $(2+1)$-ES does not make a distinction between data item level and transaction level requests. It always operates at the data item level irrespective of how the request was made. Similar to the data item level case, the $(2+1)$-ES here first determines the set of data items to be scheduled in order to serve all requests in the queue – $\bigcup \mathcal{D}_j$. It then generates a schedule with these items to maximize the utility. Since requests are served only when all requested data items are received by a client, random initial solutions are likely to disperse dependent data items (belonging to the same transaction). Hence, we modified the initialization strategy to direct the search from a favorable region in the search space. The modification creates the initial solutions by choosing random requests from the queue and sequentially laying out the data items contained in the requests on the schedule. For example, if $\mathcal{D}_1 = \{D_1, D_2, D_3\}$ and $\mathcal{D}_2 = \{D_3, D_2\}$ are two transactions on the request queue, and transaction 1 is randomly chosen under this process, then data items D_1, D_2 and D_3 will be consecutive on the initial schedule. The process generates a favorable arrangement of the data items for some of the requests. The remaining functionality of the ES is maintained as described in Section 4.3.

*5.3. ASETS**

In addition to T-EDF and T-HUF, we experiment with another heuristic called ASETS* for transaction level schedules. ASETS* is a parameter-free adaptive policy that attempts to combine the benefits of EDF and the SRPT (Shortest Remaining Processing Time) heuristics by adaptively choosing the right algorithm depending on the load in the system [23]. The algorithm is based on the argument that EDF is often useful in minimizing the tardiness (amount of delay after the deadline) of deadline based requests in a lightly loaded system, while SRPT proves to be a better choice under high loads. Under ASETS*, two separate lists are maintained, one with requests that can still make their deadlines (and ordered by their deadlines) and another with those that have already missed their deadlines (ordered by their remaining processing time). The heuristic then chooses the top request from one of the two lists, based on the total negative impact on the total tardiness of the system. Note that the idea of web transactions used in the original work is fundamentally different from that of a transaction in this work. Nonetheless, adaptive switching between EDF and SRPT can be potentially useful when transactions have low/high overlaps in the data items they request.

6. Experimental Setup

The data sets used in our experiments are generated using various popular distributions that are known to capture the dynamics of a public data access system. The different parameters of the experiment are tabulated in Table 1 and discussed below. We generate two different data sets with these parameter settings, one involving data item level requests and another involving transaction level requests. Experiments are run independently on the two data sets using the corresponding heuristics and the $(2+1)$-ES.

Each data set contains $10,000$ requests generated using a Poisson distribution with an arrival rate of

Table 1: Experiment parameters.

Parameter	Value	Comment
N	300	Number of data items
d_{min}	5KB	Minimum data item size
d_{max}	1000KB	Maximum data item size
b	120 KB/s	Broadcast channel bandwidth
m	60s/180s	Request response time mean for data item/transaction level requests
σ	20s/40s	Request response time standard deviation for data item/transaction level requests
r	5	Request arrival rate
r_T	5	Transaction size Poisson distribution rate
s	0.8	Zipf's law characterizing exponent
P	Low(1), Medium(2), High(4)	Priority levels
p	0.5	Mutation probability
Gen	1000	Number of iterations for local search and ES

r requests per second. Each request consists of an arrival time, data item number (or set of data item numbers for transaction level requests), an expected response time, and a priority level. For transaction level requests, the number of data items contained in a transaction is drawn from another Poisson distribution with rate r_T items per transaction.

We would like to point here that the number of data items (N) to be handled by a scheduler need not be very big for a particular service. Each broadcast station would typically be responsible for catering requests involving a sub-domain of the entire setup, in which case its data domain would also be comparatively smaller. The data items requested are assumed to follow the commonly used *Zipf*-like distribution [29] with the characterizing exponent of 0.8. Under this assumption, the first data item becomes the most requested item, while the last one is the least requested. Broadcast schedules can be heavily affected by the size of the most requested data item. Hence, we consider two different assignments of sizes adopted from earlier works [3, 8]. The *INC* distribution makes the most requested data item the smallest in size, while the *DEC* distribution makes it the largest in size.

$$INC : d_i = d_{min} + \frac{(i-1)(d_{max} - d_{min} + 1)}{N}$$
$$DEC : d_i = d_{max} - \frac{(i-1)(d_{max} - d_{min} + 1)}{N}$$

Expected response times are assigned from a normal distribution with mean m and standard deviation σ. The particular settings of these parameters in our data item level experiment results in expected response times to be generated in the range of $[0, 120]$s with a probability of 0.997. For transaction level requests, the mean and standard deviation are changed so that the interval changes to $[60, 300]$s. Any value generated outside the range of the intervals is changed to the lower bound of the corresponding interval.

We use three different priority levels for the requests – *low, medium,* and *high*. Numeric values are assigned to these levels such that the significance of a level is twice that of the previous one. Since the total utility is related to the priority of the requests, we make assignments from these levels in such a way that the maximum utility obtainable by serving requests from each priority level is probabilistically equal. To do so, we use a roulette-wheel selection [27] mechanism which effectively sets the probability of selecting a priority level as: $P(Low) = \frac{4}{7}$, $P(Medium) = \frac{2}{7}$, and $P(High) = \frac{1}{7}$. This ensures that no priority value is over-represented in terms of its ability to contribute to the total utility obtainable. This characteristic is required so that heuristics like HUF cannot ignore low priority requests in an attempt to maximize the total utility.

Workloads on the scheduler can be varied by either changing the request arrival rate, or the channel bandwidth. We use the latter approach and specify the bandwidth used wherever different from the default.

The local search for EDF/LS, HUF/LS, T-EDF/LS and T-HUF/LS are run for Gen number of iterations. For $(2 + 1)$-ES, the same number of

Table 2: Percentage of maximum total utility obtained by the solution methods on data item level scheduling.

	EDF	**EDF/LS**	**HUF**	**HUF/LS**	**RxW**	**SIN2**	**NPRDS**	**(2+1)-ES**
INC	83.52	88.21	96.32	**97.88**	73.71	73.87	77.01	97.26
DEC	20.72	40.24	42.13	55.51	42.04	42.46	42.87	**69.07**

iterations is chosen as the termination criteria. The number of iterations has been fixed by experimentation such that the wall-clock scheduling time is not more than 0.01s on a 2.4 GHz Pentium 4 machine with 512 MB memory and running Fedora Core 7. This is critical so that the runtime of the iterative algorithms do not impose high latencies in the decision making process. However, the non-iterative heuristics have a much lower runtime – in the order of 2 - 5 milliseconds. The performance of each method is measured at the time instance when all the requests get served. In other words, the performance of each method is given by the sum of the utilities generated by serving each request in the data set.

7. Results and Discussion

We first present the overall performance results obtained for the different solution methodologies on the data item level data set. Although, the data item level scheduling problem can be viewed as a special case of the transaction level scheduling, we observed that a method's performance can be quite different in these two problem classes.

7.1. Data Item Level Scheduling

Table 2 shows the performance in terms of the percentage of maximum total utility returned by using each of the methods. The maximum total utility is obtained when every request is served within its expected response time, in which case it attains an utility equal to its priority level. Thus, summing up the priorities of all requests gives the maximum total utility that can be obtained.

For the *INC* data size distribution, HUF, HUF/LS and ES have similar performance. Although, EDF and EDF/LS have a slightly lower performance, both methods do reasonably well. A major difference is observed in the amount of improvement gained in EDF by using local search, as compared to that of HUF. The other three heuristics demonstrate a comparatively lower achievement in total utility.

7.1.1. EDF vs. HUF

Differences arising in the performance of EDF and HUF can be explained using Fig. 6. The top row in the figure shows the utility obtained by serving a request. Clearly, the accumulation of points is mostly concentrated in the $[0, 0.5]$ range for EDF (left). For HUF (right), three distinct bands show up near the points 4, 2, and 1 on the y-axis. A high concentration of points in these regions indicate that a good fraction of the requests are served within their response time requirements. Moreover, even if the response time requirement could not be met, HUF manages to serve them with a good utility value. The figure confirms this point since the band near 0 utility in HUF is not as dense as in EDF.

The bottom row in Fig. 6 plots the utility of the requests served during a particular broadcast. We notice the presence of vertical bands in EDF (left) which shows that a good number of requests get served by a single broadcast. This is also validated by the fact that EDF does almost half the number of broadcasts as done by HUF (right). For an intuitive understanding of this observation, consider the instance when a broadcast is ongoing. Multiple new requests come in and accumulate in the queue until the current broadcast ends. Most of these requests would be for data items that are more frequently requested. When EDF generates a schedule for the outstanding requests, it gives preference to the request that is closest to the deadline, or has crossed the deadline by the largest amount. Since the queue will mostly be populated with requests for frequently requested items, chances are high that EDF selects one of such requests. Thus, when a broadcast for such an item occurs, it serves all of the corresponding requests. This explanation is invalid for HUF since preference is first given to a data item that can generate the most utility. Since the data item with highest utility may not be the most requested one, more broadcasts may be needed for HUF.

Further, when the request queue gets long, EDF's preference to requests waiting for a long time to be served essentially results in almost no utility from

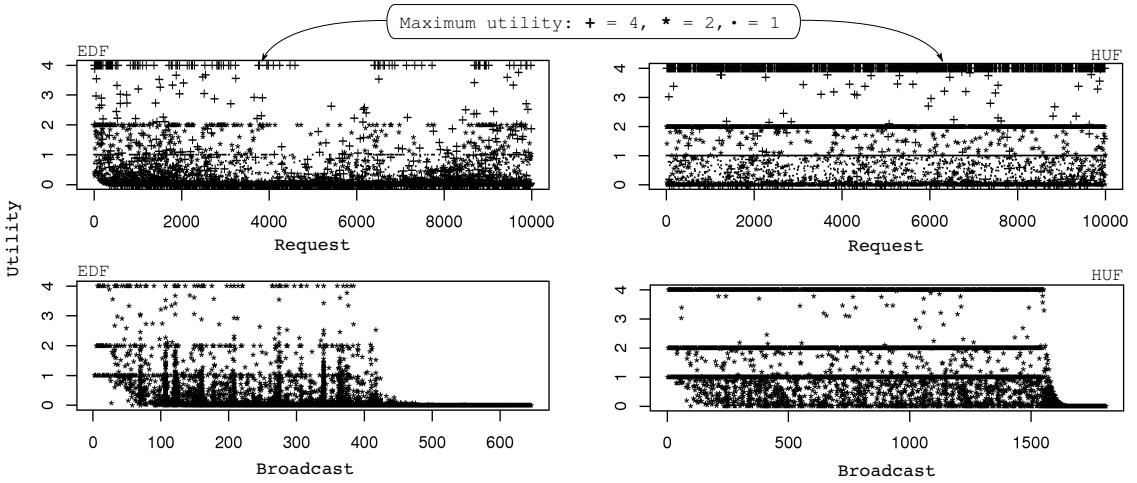


Figure 6: Utility derived by using EDF and HUF with INC data size distribution. Top: Utility obtained from each of the request for EDF (left) and HUF (right). Bottom: Utility of requests served during a broadcast for EDF (left) and HUF (right).

serving that request. If we extend this observation into the scheduling decisions taken over a long time, EDF will repeatedly schedule older and older requests. If the queue size continues to grow, this essentially results in EDF failing to generate any substantial utility after a certain point of time, as seen in Fig. 6 (bottom).

7.1.2. Impact of local search

Local search improves the EDF and HUF results for the DEC distribution up to almost between 75% and 100%. Recall that the DEC distribution assigns the maximum size to the most requested data item. If a schedule is not carefully built in this situation, there could be heavy losses in utility because other requests are waiting while the most requested data item is being broadcast. It is important that the scheduler does not incorporate the broadcast of heavy data items too frequently into its schedule and instead find a suitable trade-off with the induced utility loss. Unfortunately EDF and HUF generated schedules fail to maintain this sought balance. To illustrate what happens when local search is added, we refer to Fig. 7.

To generate Fig. 7, we enabled the local search mechanism when the 3000^{th} scheduling instance with EDF is reached. At this point, the request queue contained requests for 127 unique data items. The local search mechanism uses the 2-exchange operator to generate a new schedule. To begin with, the 2-exchange operator is applied to the EDF schedule to generate a new one. If the utility improves by more than 20 units, the new schedule

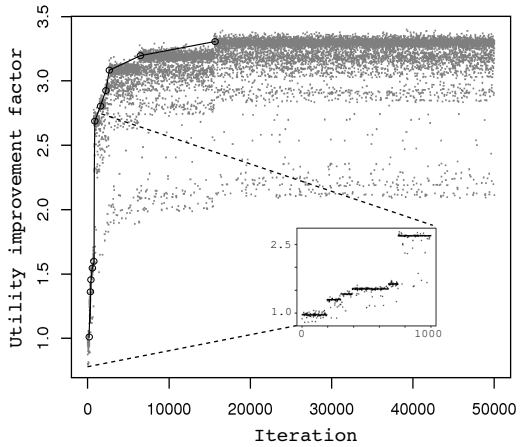


Figure 7: Improvements obtained by doing 50000 iterations of the 2-exchange operator for the EDF schedule generated during the 3000^{th} scheduling instance. The DEC data size distribution is used here. A newly obtained solution is considered better only if it exceeds the utility of the current solution by at least 20 units.

is considered for any further 2-exchange operation. The process is repeated for 50000 iterations. The plot shows the factor of improvement obtained from the EDF schedule at each iteration of the local search. The solid line joins the points where a generated schedule had an utility improvement of 20 units, or more, over the current one.

The horizontal bands in the figure illustrate the fact that the schedule space is mostly flat in structure. For a given schedule, most of its neighboring schedules (obtained by the 2-exchange opera-

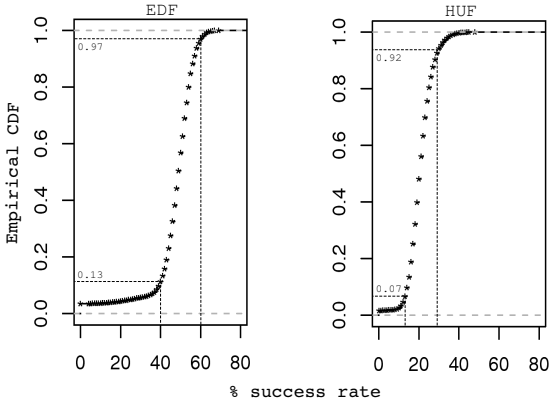


Figure 8: Empirical cumulative density function of the % success rates of performing 100 independent iterations of the 2-exchange operation in each scheduling instance with EDF (left) and HUF (right). A success means the operation resulted in a better schedule.

tor) have, more or less, equal amounts of utility associated with them. An interesting observation is that the schedule utility improves to a factor of 2.5 within the first 1000 iterations of the local search (inset Fig. 7), after which the progress slows down. This implies that as the schedules become better and better, the local search mechanism finds it difficult to generate a still better schedule (observe the long horizontal band stretching from around the 15000 to the 50000 iteration).

Since the results indicate that EDF and HUF both gain substantial improvement with local search, it can be inferred that their schedules have much room for improvement when the utility measurement is the metric of choice. This is evidenced in Fig. 8. To generate the plots, the space near an EDF (HUF) generated schedule is sampled 100 times. Each sample is obtained by applying the 2-exchange operator to the heuristic generated schedule. A success is noted if there is an increase in utility of the schedule. With the success rate (number of success/number of samples) obtained in all the scheduling instances for the 10000 requests, an empirical cumulative density function is constructed. A point (x, y) in the plot evaluates to saying that in y fraction of the scheduling instances, a better schedule is obtained 0 to x times out of the 100 independent samples taken. For EDF, about 84% (97% − 13%) of the schedules have been improved 40 to 60 times. This high success rate for a majority of the schedules generated indicate that EDF is not a good heuristic to consider in the context of utility. HUF displays a similar increase in utility, but with a relatively lower success rate. HUF schedules generate higher utilities than EDF schedules and hence the success rate is low.

Table 3: (2+1)-ES generated total raw utility value shown as *mean/standard deviation* from 30 runs. The maximum total utility that can be generated is: 17230 and 16961 units in the data item level and transaction level data sets, respectively.

	Data Item	**Transaction**
INC	16761.07/51.54	12657.14/83.62
DEC	11728.61/167.35	6910.68/113.81

The observations till this point help us make the following conclusions. The nature of the search space tells us that significant improvements can be obtained by using local search on heuristic schedules, specially when the schedule utilities are substantially lower than what can be achieved. We observe that EDF and HUF in fact generate schedules that are not difficult to improve with a single swap of the data item positions. Therefore, combining local search to both the heuristics enable us to at least "climb up" the easily reachable points in the search space.

7.1.3. HUF/LS vs. $(2+1)$*-ES*

Our justification as to why local search with an operator like 2-exchange fails after a certain point is based on the fact that these operators are limited by the number of points they can sample – the neighborhood. As schedules become better, much variation in them is required to obtain further improvement. Mutation based operators are designed to limit this variation while recombination can sample the search space more diversely [30, 31]. This is the primary motivation behind using a recombinative ES to get improved schedules. In addition, our initial experiments showed that a higher mutation rate (0.5 used here) is useful in better sampling a flat search space, specially when working with a restricted population size. Table 3 lists few dispersion measures to statistically validate the results from the ES. Standard deviation measures derived from 30 independent runs of the algorithm are less than 1% in all the scenarios.

To analyze the performance differences in HUF/LS and ES, we make the problem difficult by reducing the bandwidth to 80 KB/s. The *DEC* data size distribution is used and the broadcast frequencies for the 300 data items are noted. Fig. 9 (top) shows the frequency distribution. A clear distinction is observed in the frequencies for average

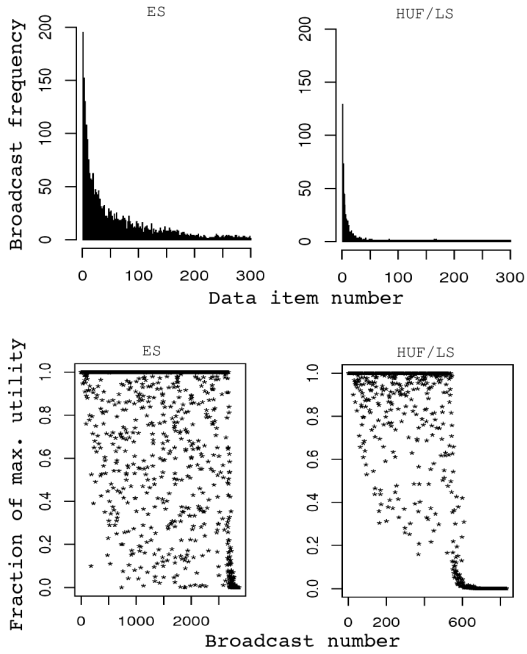


Figure 9: Top: Broadcast frequency of the N data items for ES (left) and HUF/LS (right). Bottom: Fraction of maximum utility obtained from the different broadcasts for ES (left) and HUF/LS (right). The DEC distribution is used with a 80 KB/s bandwidth.

sized data items. Recall that HUF first schedules the data item with the highest utility. However, it fails to take into account the impact of broadcasting that item on the utility that can be generated from the remaining requests. Since a majority of the requests are for the larger data items, it is highly likely that such items get scheduled more frequently. As a result, most of the bandwidth is used up transmitting heavy data items. The impact of this is not felt on small data items as they are not requested often. However, for average data items that do have a substantial presence in the requests, utility can suffer. The difference between the HUF/LS schedule and ES schedule appears at this point.

HUF schedules broadcast average sized items too infrequently, which implies that most requests for them wait for a long time before getting served. ES schedules have a comparatively higher frequency of broadcast for such data items, thereby maintaining a trade-off between the utility loss from not servicing frequently requested items faster and the utility gain from servicing average data items in a uniform manner. As can be seen from Fig. 9 (bottom), the pitfalls of the absence of this balance in HUF/LS is observed after a majority of the broadcasts has been done. HUF/LS schedules do perform better in maintaining a good fraction of the maximum utility during the initial broadcasts (notice that majority of the points are above the 0.2 limit on the y-axis prior to the 600^{th} broadcast). Much of the difference in performance arises because of the utility losses resulting after that. In contrast to that, ES schedules consistently balance losses and gains to perform well almost till the end of the last broadcast.

7.1.4. Other heuristics

The remaining three heuristics, namely RxW, SIN2 and NPRDS, employ elaborate strategies to control the size of the request queue. However, these strategies seem to have a minimal impact, if not adverse, on the performance of the heuristics. All of these heuristics achieve a comparatively lower utility level on the INC distribution, which appears to be a relatively easier problem for the other methods. Nonetheless, considering factors such as number of pending requests and data item size plays a vital role when most data items to be broadcast are comparatively larger in size. The three heuristics account for these features and achieve equal levels of utility as achieved by HUF or EDF/LS. There exists correlation between how fast the request queue can be cleared and the utility achieved in the process. Unlike EDF, these heuristics have a strategic bias towards broadcasts that can clear a larger number of requests from the queue. Local search remains a viable supplement to further improve the schedules.

7.2. Transaction Level Scheduling

Table 4 shows the performance of the solution methods on the transaction level scheduling problem in terms of the percentage of maximum utility attained. Similar to the data item level scheduling, the problem is not difficult to solve for the INC type data size distribution. Hybrid methods involving the use of local search appear to be particularly effective. The problem is comparatively difficult to solve with the DEC type distribution. Nonetheless, local search provides substantial improvement on the quality of the heuristic generated schedules in this case as well.

7.2.1. Performance on INC

The INC type distribution assigns the smallest size to the most requested data item. With this

Table 4: Percentage of maximum total utility obtained by the solution methods for transaction level scheduling.

	T-EDF	**T-EDF/LS**	**T-HUF**	**T-HUF/LS**	**ASETS***	**(2+1)-ES**
INC	66.69	**87.78**	70.92	87.68	80.09	74.18
DEC	33.47	**54.82**	27.14	50.88	14.95	40.77

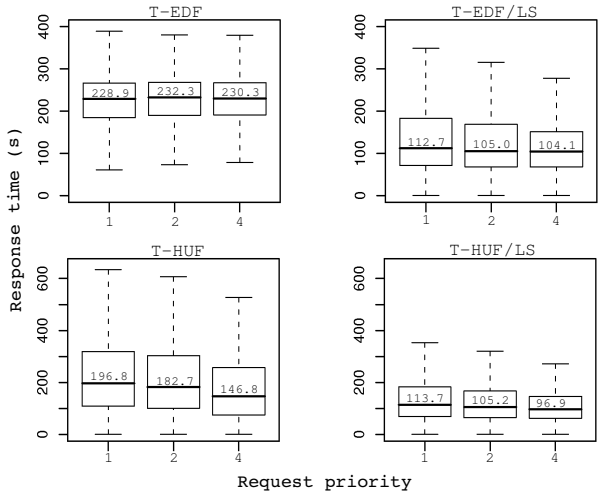


Figure 10: Whisker plots of response time for earliest deadline and highest utility heuristics on transaction level scheduling with the *INC* distribution. T-EDF fails to distinguish between requests of different priority levels. T-HUF makes the distinction on the average but response times are often high. Local search suitably modifies the schedules in both cases.

distribution, most requests have smaller data items in the requested set. Typical bandwidth values, as used in these experiments, thus allow multiple (and different) requests to be served in a relatively short span of time. Most difficulties in scheduling arise when sporadic broadcasts of bigger data items are made and the request queue grows in this duration.

Fig. 10 shows whisker plots of the response times for the earliest deadline and the highest utility heuristics, along with their local search variants, when applied on the data set with the *INC* distribution. The response time of a request is the difference in time between its arrival and the broadcast of the last remaining data item in the requested set. Note that, despite the high variance in response time, T-HUF manages a higher utility than T-EDF. The only visible factor better in T-HUF is the median value of these response times. We thus focus our attention to this statistic. Using the median response time as the metric, we see that T-EDF maintains similar values across requests of different priorities. It fails to identify that high priority requests can generate more utility and hence the data items in such requests should receive some level of preference over low priority requests. The observation is not surprising since T-EDF's efforts are restricted to the deadline requirement only. In effect, there is no distinction between two requests with the same amount of time remaining to their deadlines, but with different priorities. T-HUF makes the distinction clear and further differentiates between such requests with the added advantage of being able to identify broadcasts that might serve multiple low priority requests instead of just a single high priority request.

Although T-EDF and T-HUF's differences come from the ability to distinguish between priority levels of requests, it does not seem to be the only factor affecting the utility. Local search on these methods result in substantial improvement. As described in the hypothetical example in Section 5.1, both T-EDF and T-HUF do not take into account the effect of intermingling data items on the schedule utility. By swapping data items across the schedule, local search effectuates better exploitation of the fact that commonly requested data items are present in a significant fraction of the requests in the queue.

7.2.2. Performance on DEC

A major difference between data item level scheduling and transaction level scheduling is in the number of data items that has to be broadcast to serve the requests in the data set. With an average transaction size of 5 data items in each request, transaction level scheduling has to make 5 different broadcasts before a single request is served. Thus, heuristics in this problem must be able to exploit any overlaps between requests to be effective in utility. The problem is further complicated with the *DEC* type distribution. All requests in this case will contain at least one or more big data items as a result of the *Zipf* distribution. If broadcast of such items (taking a long time to broadcast) serves a very small fraction of the requests, then utility will most likely suffer in the long run. Further, average or smaller sized data items will be requested

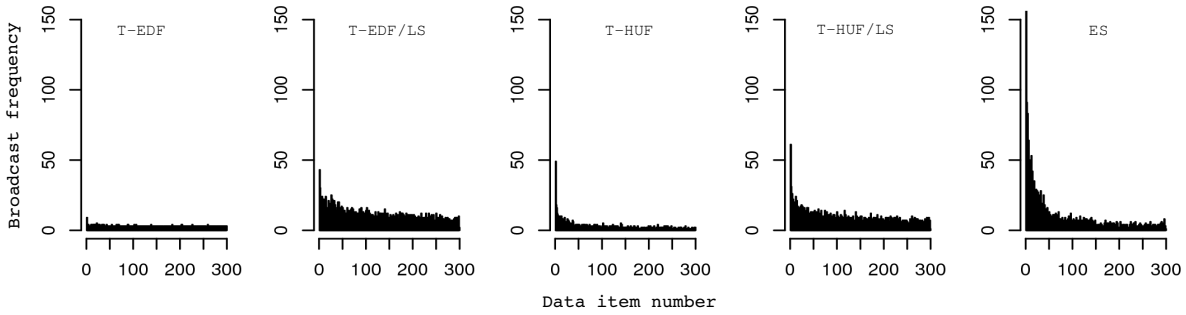


Figure 11: Broadcast frequency of the N data items for transaction level scheduling in the DEC distribution. Distribution of average and below-average sized data items reaches a uniform trend for high utility solution methods.

with a less frequency, often by requests separated far apart in the time line. Once again, utility will suffer if a heuristic waits for a long time to accumulate requests for such data items with the objective of serving multiple such requests with a single broadcast. Thus, a heuristic must be able to balance these two aspects of broadcasting when working on transaction level scheduling with the DEC distribution.

Fig. 11 shows the broadcast frequency of the different data items on this problem. The frequency distribution is very dissimilar than in the case of data item level scheduling. T-EDF, which performs better than T-HUF here, manages to maintain an almost uniform distribution. In fact, if the first 20% of the bigger data items are not considered, all methods show a similar trend. The uniform distribution is an extreme form of the balance we seek as indicated above. T-HUF encounters a problem when seeking this balance and results in a marginal drop in performance as compared to T-EDF. In transaction level scheduling, requests which overlap significantly with parts of other requests are the ones that can generate the highest utility. The overlap mostly occurs in the frequently requested data items. T-HUF schedules such requests first without taking into account that delaying a broadcast for a commonly requested item can actually help accumulate more requests interested in the same data item. Unless a new request has better overlap features than the ones already existing in the queue, the T-HUF schedule undergoes very small changes and virtually remains consistent, for a certain period of time. What disrupts the consistency is the accumulation of enough new requests to change the overlap features altogether. However, multiple data items (often the bigger ones) have already been broadcast by this time and T-HUF is needed to schedule another broadcast for them. In the case of T-EDF, this effect is overpowered to some extent by the deadline requirement. As and when a new request with a smaller deadline, and preferably with a lesser overlap with the existing requests arrive, T-EDF immediately gives preference to it. As a result, broadcast times of data items in the existing schedule is delayed, which helps serve more requests now. The frequency of broadcast of commonly requested data items in T-EDF and T-HUF corroborates our justification.

Performance improvements in these heuristics when aided by local search is mostly attributable to the higher frequency of broadcasts for average and smaller sized data items. Recall that a request is served only when all data items of interest are received by it, irrespective of the order of retrieval. Consider a request interested in the data items D_1, D_2, and D_3, with D_1 being a commonly requested item under the DEC distribution. Next, consider the two schedules – $D_1 < D_2 < D_3$ and $D_3 < D_2 < D_1$. In both cases, the client making the request completes retrieval of the data items at the same time instance. However, the latter schedule has the advantage of delaying the broadcast of D_1, which in effect improves the chances of it serving newly accumulated requests as well. Exploiting such strategies is not possible by T-EDF and T-HUF unless aided by an exchange operator of the kind present in the local search variant. The side effect of this is that infrequently requested data items will be more often broadcast. This is clearly visible in T-EDF/LS and T-HUF/LS.

7.2.3. Dynamics of local search

The ability to exploit the overlap features of requests makes local search a promising aid to the heuristic generated schedules in both data item

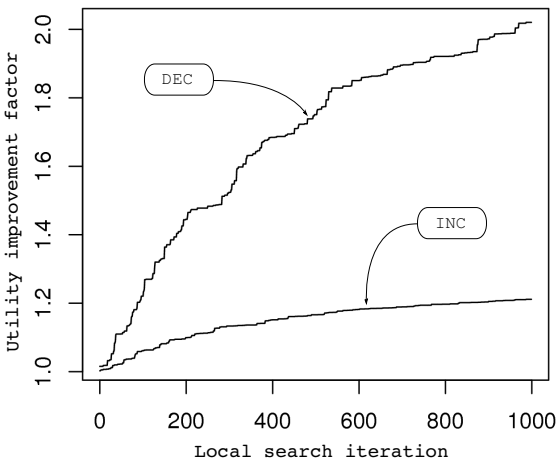


Figure 12: Improvements obtained by local search in T-EDF/LS during the 5000^{th} scheduling instance. Improvements are minor when the T-EDF schedule is already "good" (INC distribution). Utility shows an increasing trend at the end of the 1000th iteration of the local search for the DEC distribution.

level and transaction level scheduling. Recall from Fig. 7 that the local search is most effective when schedules can be easily improved by adopting a hill-climbing approach. The search space becomes more and more flat as better schedules are discovered. Often, reaching this flat region is not difficult in data item level scheduling, both for the INC and DEC distribution. However, the dynamics are somewhat different in transaction level scheduling.

Fig. 12 shows the improvement gained over a T-EDF generated schedule during the 5000^{th} scheduling instance in T-EDF/LS. The plot depicts the progress of local search in its search for a better schedule. The T-EDF generated schedule is closer to a flat region for the INC distribution than for the DEC distribution. Hence, improvements are often slow in the INC distribution. However, observe that for the DEC distribution, at the end of the 1000^{th} iteration of local search, the improvements are still showing an increasing trend. This indicates that, given more number of iterations, local search can continue to easily improve the schedules before hitting the flat regions. Given the diverse number of factors determining the utility of schedules in the DEC distribution, it is not surprising to see that T-EDF generated schedules are far from being optimal. Besides, a hill-climbing strategy like local search can effectively exploit the dynamics of the search space to adjust these schedules to bring them closer to the optimal. The only hindrance we face in doing so is the right adjustment of the number of iterations without imposing a bottleneck in the running time of the scheduler. We provide some suggestions on how this can be done in a later subsection.

7.2.4. Performance of $(2+1)$*-ES*

Our primary motivation behind using $(2+1)$-ES in data item level scheduling is to enable a diverse sampling of the search space when a schedule's utility no longer shows quick improvements. However, for transaction level scheduling, other factors start to dominate the performance of this method.

Transaction level scheduling leads to an explosion in the size of the search space. For data item level scheduling, a request queue of size n with requests for distinct items has a schedule search space of size $n!$, whereas in transaction level scheduling with k items per transaction, the search space can become as big as $(kn)!$. A bigger search space not only requires more exploration, but can also inject a higher possibility of prematurely converging to a local optima. Fig. 13 depicts the exploration with local search and the $(2+1)$-ES. In order to generate the plots, we ran T-EDF until the 5000^{th} request arrives. A schedule for the requests in the queue at this point is then generated by using T-EDF/LS and $(2+1)$-ES ran for 5000 iterations each. This guarantees that both methods are operating on the same queue (same search space) and for sufficient number of iterations. Utility improvements by the ES starts stagnating after the 2000^{th} iteration, suggesting convergence towards a local optima. Local search, on the other hand, has a better rate of improvement before starting to stagnate. This is visible for both INC and DEC distributions.

The premature convergence in $(2+1)$-ES is mostly due to the loss in *genetic diversity* of the solutions. As more and more recombination takes place, the two parents involved in the method starts becoming more and more similar, and finally are unable to generate a diverse offspring. This phenomenon is what typically identifies the convergence of the method. However, given the bigger search space and the small population (two) involved in exploring it, the phenomenon results in the method getting trapped in the local optima present across the space. It is suggestive from the broadcast frequency of different data items (Fig. 11) that, on the overall, the method failed to balance the broadcast of different sized data items for

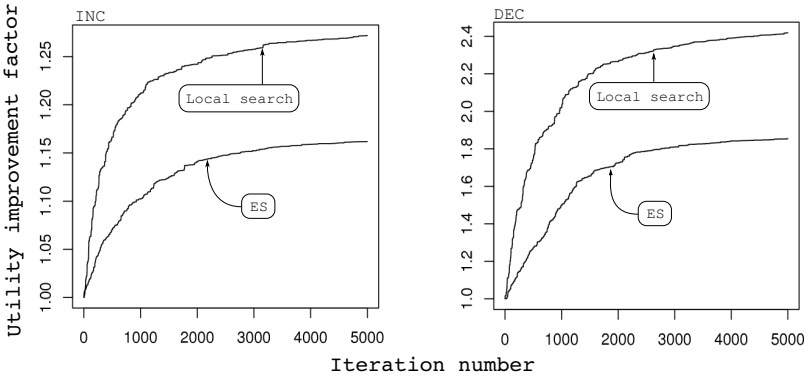


Figure 13: Improvements obtained by local search and ES in the schedule generated by T-EDF during the 5000^{th} scheduling instance. For transaction level scheduling, local search performs better in avoiding local optima compared to $(2+1)$-ES.

the *DEC* distribution. Changing the μ and λ parameters, as well as the mutation probability of the method usually helps resolve such premature convergence. However, one should keep the real time runtime constraints into consideration while experimenting with the parameters. Local search, on the other hand, does not face such a problem. Ideally, given enough number of iterations, a hill-climbing approach can be made resilient against getting trapped in a local optimum using random restarts.

*7.2.5. Performance of ASETS**

The ability to adaptively choose between EDF and SRPT appears to have a positive impact on the total utility achievable by a heuristic. The strategy works considerably well on the *INC* data size distribution, with global utility levels surpassing that of a naive heuristic like T-EDF or T-HUF. However, the same strategy has a much negative impact when the data size distribution is reversed. In this case, ASETS* fails to generate almost any utility in the broadcast data. Note that ASETS* does not distinguish between the different data items that form a transaction, but only considers the time required to broadcast all the remaining items in a transaction. This disables the algorithm from identifying existing overlaps between transactions. Exploiting these overlaps becomes crucial in a broadcast scenario where most items will occupy the broadcast bandwidth for a relatively longer extent of time. This result further corroborates the claim that scheduling for multiple data items is not a simple extension of the data item level counterpart.

7.3. Scheduling Time

The number of generations allowed to local search, or ES, can affect the quality of solutions obtained and the time required to make scheduling decisions. In our experiments, this value is set so that an average request queue can be handled in a small amount of time. However, the average queue size will greatly vary from problem to problem, often depending on the total number of data items served by the data source. In such situations, it may seem difficult to determine what a good value for the number of iterations should be. Further, in a dynamic environment, the average queue length itself may be a varying quantity. Nonetheless, one should keep in mind that scheduling decisions need not always be made instantaneously. The broadcast time of data items vary considerably from one to the other. The broadcast scheduled immediately next cannot start until the current one finishes. This latency can be used by a scheduler to continue its search for better solutions, specially with iterative methods like a local search or an ES.

8. Conclusions

In this paper, we address the problem of time critical data access in wireless environments where the time criticality can be associated with a QoS requirement. To this end, we formulate an utility metric to evaluate the performance of different scheduling methods. The earliest deadline first (EDF) and highest utility first (HUF) heuristics are used in two problem domains – data item level scheduling and transaction level scheduling. In the data item level domain, our initial observation on their performance conforms to the speculation that

HUF performs better since it takes into consideration the utility of requests while making scheduling decisions. Also, strategies adopted in elaborate heuristics such as RxW, SIN-α and NPRDS have a positive impact on certain classes of problem only. Further analysis of the nature of the scheduling problem shows that local search can be a promising supplement to a heuristic.

The observations drawn from this understanding of the behavior of local search aided heuristics enable us to propose an evolution strategy based search technique that provides more variance to a simple local search. The utility induced by such a technique surpasses that of both EDF and HUF, and their local search variants. This result also shows that search based optimization techniques are a viable option in real time broadcast scheduling problems.

The transaction level domain appear to be a relatively harder problem to solve. Balancing the broadcast frequency of data items is important here and we observe that T-EDF performs better in doing so as compared to T-HUF. Local search still remains a promising candidate to boost the performance of these heuristics. For certain data size distributions, local search improvements can continue if allowed to run outside the runtime restrictions set in our experiments. Better strategies to trigger the scheduler is thus required. Transaction level scheduling also has different search space dynamics, often with an explosive size and the presence of local optima. The evolution strategy method is affected by this, often leading to lower utility schedules than local search. Further experimentation is required in this direction to see how the parameters of the ES can be changed to avoid susceptibility towards such situations.

From a utility standpoint, we need to explore the option of designing heuristics that pay special attention to factors like broadcast frequency and loss-gain trade-off during scheduling decisions. The validity of a broadcast is to be taken into account when requests involve multiple data items which may undergo regular updates. Timely delivery of a data item then has to consider a validity deadline as well. Application usage characteristics can also be incorporated while making utility based scheduling decisions. This is particularly useful since different applications may have different time criticality requirements on the data, which in turn implies different time-utility functions. The overall QoS in the system will then depend on how the broadcast schedule incorporates knowledge about the different utility functions.

Acknowledgments

This work was partially supported by the U.S. Air Force Office of Scientific Research under contract FA9550-07-1-0042. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the U.S. Air Force or other federal government agencies.

References

[1] W. de Jonge, Kilometerheffing op basis van elektronische aangifte, Informatie (2003) 22–27.

[2] S. Jiang, N. H. Vaidya, Scheduling Data Broadcasts to "Impatient" Users, in: Proceedings of the 1st ACM International Workshop on Data Engineering for Wireless and Mobile Access, 1999, pp. 52–59.

[3] V. C. Lee, X. Wu, J. K. Ng, Scheduling Real-Time Requests in On-Demand Data Broadcast Environments, Real-Time Systems 34 (2) (2006) 83–99.

[4] J. Xu, X. Tang, W.-C. Lee, Time-Critical On-Demand Data Broadcast: Algorithms, Analysis and Performance Evaluation, IEEE Transactions on Parallel and Distributed Systems 17 (1) (2006) 3–14.

[5] C. Su, L. Tassiulas, Broadcast Scheduling for Information Distribution, in: Proceedings of the INFOCOM '97, 1997, pp. 109–117.

[6] S. Acharya, 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, 1998, pp. 43–54.

[7] D. Aksoy, M. Franklin, RxW: A Scheduling Approach for Large-Scale On-Demand Data Broadcast, IEEE/ACM Transactions on Networking 7 (6) (1999) 846–860.

[8] S. Hameed, N. H. Vaidya, Efficient Algorithms for Scheduling Data Broadcast, Wireless Networks 5 (3) (1999) 183–193.

[9] K. C. K. Lee, W. Lee, S. Madria, Pervasive Data Access in Wireless and Mobile Computing Environments, Wireless Communications and Mobile Computing (2008) 25–44.

[10] P. Xuan, S. Sen, O. Gonzalez, J. Fernandez, K. Ramamritham, Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments, in: Proceedings of the 3rd IEEE Real-Time Technology and Applications Symposium, 1997, pp. 38–48.

[11] K. Lam, E. Chan, J. C. Yuen, Approaches for Broadcasting Temporal Data in Mobile Computing Systems, Journal of Systems and Software 51 (3) (2000) 175–189.

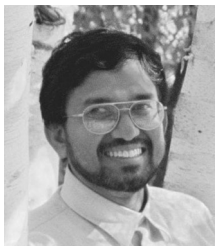
[12] J. Fernandez, K. Ramamritham, Adaptive Dissemination of Data in Time-Critical Asymmetric Communication Environments, Mobile Networks and Applications 9 (5) (2004) 491–505.

[13] X. Wu, V. C. Lee, Wireless Real-Time On-Demand Data Broadcast Scheduling with Dual Deadlines, Journal of Parallel and Distributed Computing 65 (6) (2005) 714–728.
[14] J. Chang, T. Erlebach, R. Gailis, S. Khuller, Broadcast Scheduling: Algorithms and Complexity, in: Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms, 2008, pp. 473–482.
[15] B. Ravindran, E. D. Jensen, P. Li, On Recent Advances in Time/Utility Function Real-Time Scheduling and Resource Management, in: Proceedings of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2005, pp. 55–60.
[16] E. Jensen, C. Locke, H. Tokuda, A Time Driven Scheduling Model for Real-Time Operating Systems, in: Proceedings of the 6th IEEE Real-Time Systems Symposium, 1985, pp. 112–122.
[17] G. Buttazzo, M. Spuri, F. Sensini, Value vs. Deadline Scheduling in Overload Conditions, in: Proceedings of the 16th IEEE Real-Time Systems Symposium, 1995, pp. 90–99.
[18] H. Wu, U. Balli, B. Ravindran, E. D. Jensen, Utility Accrual Real-Time Scheduling Under Variable Cost Functions, in: Proceedings of the 11th IEEE Conference on Embedded and Real-Time Computing Systems and Applications, 2005, pp. 213–219.
[19] R. Dewri, I. Ray, I. Ray, D. Whitley, Optimizing On-Demand Data Broadcast Scheduling in Pervasive Environments, in: Proceedings of the 11th International Conference on Extending Database Technology, 2008, pp. 559–569.
[20] Y. Chehadeh, A. Hurson, M. Kavehrad, Object Organization on a Single Broadcast Channel in the Mobile Computing Environment, Multimedia Tools and Applications 9 (1) (1999) 69–94.
[21] A. R. Hurson, Y. C. Chehadeh, J. Hannan, Object Organization on Parallel Broadcast Channels in a Global Information Sharing Environment, in: Proceedings of the 19th International Performance, Computing, and Communications Conference, 2000, pp. 347–353.
[22] J.-L. Huang, M.-S. Chen, Dependent Data Broadcasting for Unordered Queries in a Multiple Channel Mobile Environment, IEEE Transactions on Knowledge and Data Engineering 16 (9) (2004) 1143–1156.
[23] S. Guirguis, M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, K. Pruhs, Adaptive Scheduling of Web Transactions, in: Proceedings of the 25th International Conference on Data Engineering, 2009, pp. 357–368.
[24] C. D. Locke, Best-Effort Decision Making for Real-Time Scheduling, Ph.D. thesis, Carnegie Mellon University (1986).
[25] H. G. Beyer, H. P. Schwefel, Evolution Strategies: A Comprehensive Introduction, Natural Computing 1 (2002) 3–52.
[26] I. Rechenberg, Evolutionsstrategie: Optimierung technischer Systemenach Prinzipien der biologischen Evolution, Ph.D. thesis, Technical University of Berlin (1970).
[27] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.
[28] T. Starkweather, S. McDaniel, C. Whitley, K. Mathias, D. Whitley, A Comparison of Genetic Sequencing Operators, in: Proceedings of the 4th International Conference on Genetic Algorithms, 1991, pp. 69–76.
[29] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web Caching and Zipf-Like Distributions: Evidence and Implications, in: Proceedings of the IEEE INFOCOM '99, 1999, pp. 126–134.
[30] H. G. Beyer, An Alternative Explanation for the Manner in which Genetic Algorithms Opearate, BioSystems 41 (1997) 1–15.
[31] J. Holland, Hidden Order: How Adaptation Build Complexity, Basic Books, 1995.

Vitae

Rinku Dewri is an Assistant Professor in the Computer Science Department at University of Denver. He obtained his Ph.D. in Computer Science from Colorado State University. His research interests are in the area of information security and privacy, risk management, data management and multi-criteria decision making. He is a member of the IEEE and the ACM.

Indrakshi Ray is an Associate Professor in the Computer Science Department at Colorado State University. She obtained her Ph.D. from George Mason University. Her research interests include security and privacy, database systems, e-commerce and formal methods in software engineering. She served as the Program Chair for SACMAT 2006 and IFIP WG 11.3 DBSEC 2003. She has also been a member of several program committees such as for EDBT, SACMAT, ACM CCS and EC-Web. She is a member of the IEEE and the ACM.



Indrajit Ray is an Associate Professor in the Computer Science Department at Colorado State University. His main research interests are in the areas of computer and network security, database security, security and trust models, privacy and computer forensics. He is on the editorial board of several journals, and has served or is serving on the program committees of a number of international conferences. He is a member of IEEE, IEEE CS, ACM, ACM SIGSAC, IFIP WG 11.3 and IFIP WG 11.9.

Darrell Whitley is Professor and Chair of the Computer Science Department at Colorado State University. He also currently serves as Chair of ACM SIGEVO. He was chair of the governing board for International Society for Genetic Algorithm from 1993 to 1997, and was Editor-in-Chief of the MIT Press journal Evolutionary Computation from 1997 to 2002.