Optimizing On-Demand Data Broadcast Scheduling in Pervasive Environments

Rinku Dewri, Indrakshi Ray, Indrajit Ray and Darrell Whitley
Department of Computer Science
Colorado State University
Fort Collins, CO 80523, USA
{rinku,iray,indrajit,whitley}@cs.colostate.edu

ABSTRACT

Data dissemination in pervasive 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 pervasive environments where the utility of the data is an equally important criterion as its timeliness. Missing the deadline reduces the utility of the data but does not make it zero. In this work, we address the problem of scheduling on-demand data broadcasts with 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 the 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 surpassing the requirement of a heuristic in the process. An evolution strategy based stochastic hill climber is investigated in this context.

1. INTRODUCTION

Recent advances in wireless communication technology is increasingly making the dream of pervasive computing a reality. Pervasive computing involves a network of portable computing devices so thoroughly embedded in our day-to-day work and personal life that their existence becomes difficult to perceive altogether. The devices interact with each other and with other computing devices 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 pervasive data access a challenging problem, but also identifies "optimal resource allocation" as one of the fundamental research problems in this domain.

A pervasive environment encompasses both peer-to-peer and client-server modes of data dissemination. For example, a typical pervasive health care system involves multiple sensor nodes disseminating data on the monitored life signs of a patient to a personal digital assistant carried by the serving health personnel. Data communication follows a peer-to-peer architecture in such frameworks. On the other hand, a pervasive 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 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 paper is directed towards data access issues in such pervasive environments.

The quality of service is an important facet in such pervasive data access. Consider the following example application – a traffic data service monitors the traffic in a large city and provides various routing services to drivers to avoid roadblocks, construction delays, congestions, accidents etc. The server gets real time traffic data via thousands of sensors spread throughout the city. Drivers request this service with devices equipped with GPS navigation units. The requests arrive at the server with various priority levels and soft deadlines. Let us assume that at some point there is a major traffic gridlock within the city and the traffic server gets thousands of re-routing requests from users. While these requests are queued up at the server, another request comes from a VIP's convoy with a high priority and deadline. At this time, the server needs to determine how to schedule this request. Pre-empting others may enable the server to meet the timeliness of this latest request. However, serving some or all of the earlier requests has the greater utility of clearing up the gridlock earlier.

In this example, the different data that the server needs to serve is associated with different utility values. Moreover, 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). Although users, who hardly realize the bottlenecks in the information infrastructure, would like to have their requests served at the earliest, it is reasonable to assume that data still carries some utility to it even if received after a specified deadline. Such an assumption enables data broadcasts to be tailored in such a way that total utility associated with a broadcast is maximized, thereby helping maintain a certain QoS level in the underlying network. In this work, we propose a dynamic scheduler that tries to maximize the overall utility of servicing requests and at the same time tries to serve as many requests in a timely manner as possible. The setup is a wireless broadcast environment as in pervasive computing applications.

Wireless broadcast mechanisms have been extensively investigated earlier. However, very few of these research 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* [9, 14, 15, 16, 18, 28, 29]. 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, as in many pervasive environments, 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 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 contribution in this paper is the development of a scheduler that is suitable for on-demand broadcasts of soft deadline data in pervasive environments. The scheduler reorganizes the broadcast queue when new requests arrive with the goal of maximizing the overall utility of all pending requests. We begin with an attempt to understand the nature of the underlying search space, and argue that traditional heuristics usually generate solutions in a worse part of this space w.r.t a given global utility measurement. We explore "local search" as an option to boost the performance of these solutions and provide arguments as to why the option is viable in a real time setting. The observations allow us to propose a light weight stochastic hill climber that surpasses the performance of a heuristic, and explicitly searches the space of schedules to maximize utility. We believe that the proposed method provides insights into better broadcast mechanisms which often clear our understanding for better heuristic design.

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 and the experimental setup are described in Section 4 and Section 5 respectively. Results and discussions from the experiments are summarized in Section 6. Finally, Section 7 concludes the paper.

2. RELATED WORK

Data broadcasting has been extensively studied in the context of wireless communication systems. Su and Tassiulas [24] study the problem in the context of access latency and formulate a deterministic dynamic optimization problem, the solution to which provide a minimum access latency schedule. Their experimental results show that the mean response times in push-based and on-demand broadcasts become similar as the request generation rate increases. Acharya and Muthukrishnan propose the *stretch* metric [2] to account for differences in service times arising in the case of variable sized data items. Their work identifies that maintaining a balance between local and global performance is a key factor in on-demand broadcasting environments. To this effect, they propose the MAX heuristic to optimize the worst case stretch of individual requests. Another attempt to balance individual and overall performance is seen in the work by Aksoy and Franklin [3]. Their RxW heuristic 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 adapted a *packet fair queuing* algorithm to the domain [11]. Their approach exploit the similarities in the two problem classes and give an efficient algorithm to solve the problem. Lee et al. provide a survey of these techniques [17] 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 request is presented by Xuan et al. [30]. Earliest deadline based on-demand scheduling is the heuristic of choice in this seminal work. Jiang and Vaidya address the issue by considering broadcast environments where clients do not wait indefinitely to get their request served [14]. They model the user impatience as an exponential distribution and propose the SRM algorithm to minimize the mean waiting time, which in turn maximizes the expected service ratio. Lam et al. look at the time criticality factor from the perspective of temporal data validity [16]. Their approach assigns an *absolute validity interval* to determine the refresh frequencies of data items in order to keep the cache status of the items updated using broadcast strategies. Fernandez and Ramamritham propose a hybrid broadcasting approach to minimize the overall number of deadlines missed [9]. Their adaptive model takes into consideration the data and time-criticality requirements to determine periodic and on-demand schedules for data items. Kim and Chwa present theoretical results on the competitive ratios of scheduling algorithms working with time constrained requests [15]. Temporal constraints on data are revisited by Wu and Lee with the added complexity of request deadlines [28]. Their $RDDS$ algorithm assigns a priority level to each requested data item based on the number of requests for it, the effective deadline and the size of the item, and broadcasts the item with the highest priority. Xu et al. propose the SIN-α algorithm to minimize the request drop rate [29]. However, their approach do 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 [18].

Several shortcomings of using a strict deadline based sys-

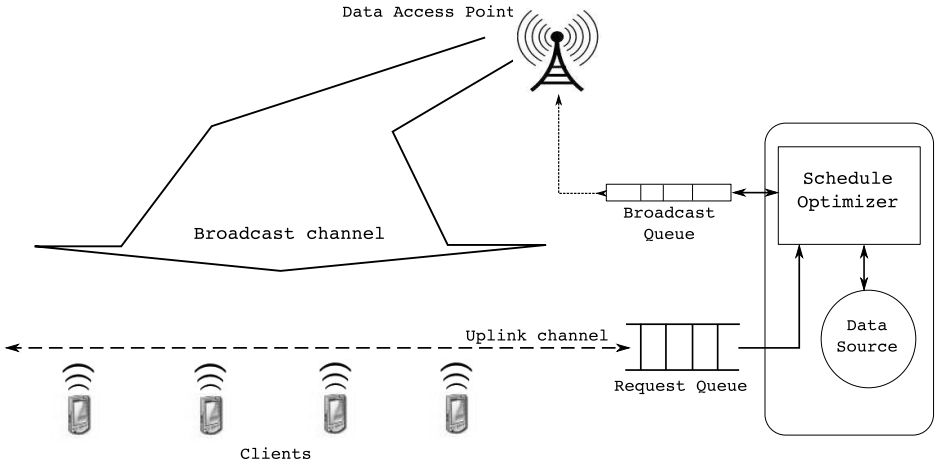

Figure 1: Typical on-demand broadcast architecture in a pervasive environment.

tem are discussed by Ravindran et al. [21] in the context of real-time scheduling and resource management. A deadline is usually a linear-valued expression that fails to distinguish between urgency and importance. Although time/utility functions in data broadcast scheduling have been ignored for some time now, the idea has been extensively researched in other real-time scheduling domains. 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 [13]. They introduce the idea of $Time - Utility\ Functions$ to capture the semantics of soft time constraints which 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 done by Buttazzo et al. [7]. Wu et al. study a task scheduling problem where utility is considered a function of the start time of the task [27]. Similar studies [8, 19, 25] performed on utility accrual in task scheduling problems show that heuristics designed to cater to the deadline requirement alone are not sufficient, and care should be taken to address any non-linear characteristics of the time-utility dependencies.

Although the different problem classes in scheduling have similarities in them, the idea of multiple requests getting served by a single broadcast make the data broadcast scheduling domain somewhat different. We believe time-utility functions are a better alternative to hard deadline specifications of data requests since they allow better generalization of the time constraints involved. Thereby, we introduce the notion of utility accrual to a data broadcast environment and explore the issues generated thereof.

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 multiple clients interested in the same data item. *Push-based* architectures broadcast commonly accessed data at regular intervals, depending on a well known access pattern, and in the process removes the requirement of a client actually sending the request to the server. Contrary to this, *on-demand* architectures allow 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. Broadcast scheduling in this context is the problem of determining the order in which data items should be broadcast so that more clients are served at a time within an acceptable quality requirement.

Data access in pervasive 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 stresses on the fact that the requested data is expected within a specified time window; failure to do so would result in an utility loss. Given the immense number of requests that may arrive at such a data broadcast server, it is often not possible to serve all requests in a timely manner. 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 pervasive environment. Various 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. 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 single channel assumption is not critical to our work, and changes in approach will be mentioned wherever appropriate.

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

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 $(t_{j,ser} - t_{j,arr})$. 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) [13]. 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,

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

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.

$$\alpha_j = \frac{\ln 0.5}{R_j} \quad (4)$$

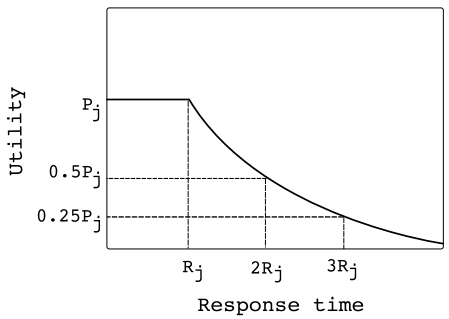

Figure 2: Utility of serving a request.

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 the 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 can have the same first component, i.e. $D_j = D_k$, for $j \neq k$; this implies that two different requests are 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 broadcasted is dependent 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 broadcasted 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 then maximize this global utility measured at any instance.

4. SOLUTION METHODOLOGY

In the previous section, we identified that a solution methodology which can maximize the global utility is desired for the broadcast problem under study. However, it is often difficult to anticipate the incoming data access requests in a dynamic environment. Besides, an ongoing broadcast cannot be interrupted to accommodate higher utility from new requests. This restricts a solution methodology to focus on the current request queue only and make scheduling decisions that yield higher utility from the generated schedule. It can only be expected that a higher global utility measure would be obtained in the process.

Another constraint is the time factor involved in making a

scheduling decision. Data requests usually arrive more frequently than they can be served, which in turn lead 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.

Moreover, the assumption that a heuristic driven method is most appropriate in real time decision making can be flawed. This is specially true when the search space involved is well understood and specialized optimization methods can be devised to exploit the dynamics of the search space. In such situations, even a suboptimal solution generated by a carefully crafted optimization technique could be better than a heuristic based solution. In the following few subsections, we present the two heuristic driven methods and a evolution strategy based optimization technique chosen for our experiments.

4.1 Heuristics

For the purpose of this study, we use two heuristics – *Earliest Deadline First* (EDF) and *Highest Utility First* (HUF) – which takes into account the time critical nature of a data request.

EDF starts off by first scheduling the data item which 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.

4.2 Heuristics with local search

As mentioned earlier, the emphasis in this paper is understanding 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) [5, 22] are a class of stochastic search methods based on computational models of adaptation and evolution. They were first suggested by Rechenberg during the late sixties. Most of the earlier work in ES did not present them as function optimizers, but rather as rules for automatic design and analysis of consecutive experiments to suitably drive a flexible system to its optimal setting.

Evolution strategies 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 [10], 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 nature 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. In the presence of multiple channels (T say), a similar encoding can be obtained by using -1 as a delimiter between the schedules in different channels (Fig. 4) [20]. 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.

```
Channel: |--1--|  |-2-|   ....   |--T--|
Encoding: 3 1 4 -1 2 6 -1 .... -1 8 5 9
```

Figure 4: Solution encoding for multiple channels.

4.3.2 Syswerda recombination

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 [23]. We employ the Syswerda 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. For a broadcast schedule, the relative ordering of data items in the schedule affect the time instance when a particular request gets served and hence influence the overall utility of the schedule.

```
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 5: Syswerda recombination.

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. 5, 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

The elementary mutation operators define 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 [5]. Insertion based mutation selects two random positions in a sequence and the element at the first chosen position is migrated to appear after the second chosen position (Fig. 6).

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 did not want to add the complexity

```
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 6: Mutation using the insertion operator.

of determining an EDF or HUF schedule to the search process, and hence generated the initial solutions x and y randomly. 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. EXPERIMENTAL SETUP

The data set used in our experiments is generated using various well known distributions that are known to capture the dynamics of a public data access system quite well. The various parameters of the experiment are tabulated in Table 1 and discussed below.

The data set contains $10,000$ requests generated using a Poisson distribution with an arrival rate of r requests per second. Each request consists of an arrival time, data item number, an expected response time, and a priority level.

The data items requested are assumed to follow the commonly used *Zipf*-like distribution [6] 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 to the data items from existing literature [11, 18]. The *INC* distribution makes the most requested data item the smallest in size, while the *DEC* distribution makes it the largest in size.

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

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

Expected response times are assigned from a normal distribution with mean m and standard deviation σ. The particular settings of these parameters in our experiment results in expected response times to be generated in the range of $[0, 120]$s with a probability of 0.997. A zero value generated using this distribution implies that the request is expected to be served immediately. Of course, because of the normal distribution, such requests are very rare to occur. Any negative value generated by the distribution is changed to zero.

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 [10] mechanism which effectively sets the probability of selecting a priority level as: $P(Low) = \frac{4}{7}$, $P(Medium) = \frac{2}{7}$, and $P(High) = \frac{1}{7}$.

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	Request response time mean
σ	20s	Request response time standard deviation
r	5	Request arrival 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

Table 1: Experiment parameters.

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

The local search for EDF/LS and HUF/LS are run for Gen number of iterations. For $(2+1)$-ES, the same number of iterations is chosen as the termination criteria. The number of iterations has been fixed such that the scheduling time is not more than 0.01s (on a 2.4 GHz Pentium 4 with 512 MB memory) when the request queue contains requests for around 150 unique data items on the average. 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 of the requests in the data set.

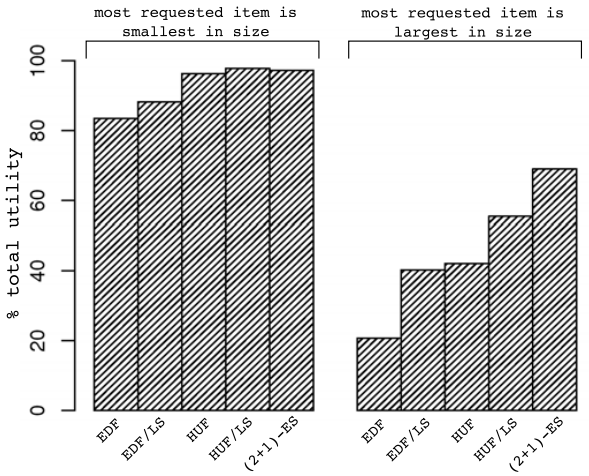

Figure 7: Percentage of maximum total utility obtained by the solution methods. The utility obtainable suffers when the most requested data item is the largest in size.

6. RESULTS AND DISCUSSION

We first present the overall performance results obtained for the five different solution methodologies. Fig. 7 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 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 the methods do reasonably well. A major difference is observed in the amount of improvement gained by EDF by using local search, as compared to that of HUF. This is because EDF does not take into consideration the utility factor of requests and hence performing a local search based on utility results in a substantial level of improvement. HUF does reasonably well in creating the initial schedules; hence local search does not provide significant additional improvement. Further explanation on this issue is presented later.

6.1 EDF vs. HUF

Differences arising in the performance of EDF and HUF can be explained using Fig. 8. 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 as the band near 0 utility in HUF is not as dense as in EDF.

The bottom row in Fig. 8 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 which 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 cor-

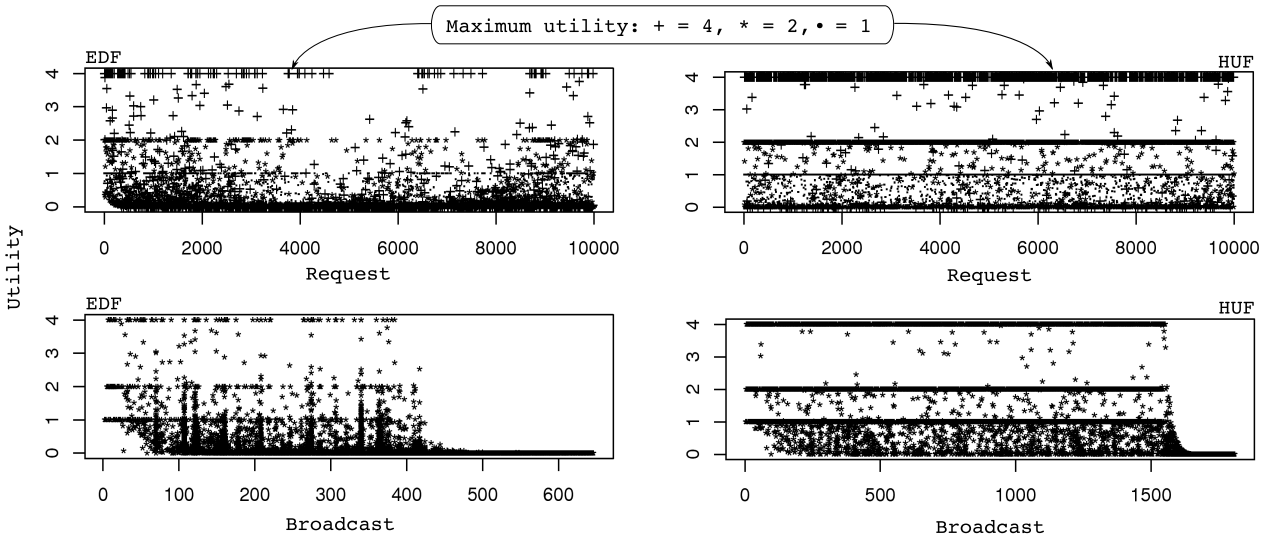

Figure 8: Utility derived by using EDF and HUF with INC data size distribution. Top: Utility obtained from each of the request for EDF (left) and HUF (right). Bottom: Utility of requests served during a broadcast for EDF (left) and HUF (right).

responding 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 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. This is clearly visible in Fig. 8 (bottom) as the early drop in the utility level of broadcasts to zero. The vertical bands in EDF suggest that the request queue size did grow to a size where a single broadcast took care of multiple requests.

6.2 Impact of local search

The impact of performing the local search improves the EDF and HUF results for the DEC distribution – up to almost 75% to 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. 9.

To generate Fig. 9, 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 use the 2-exchange operator to generate a new schedule. To start with, the 2-exchange operator is applied to the EDF schedule to

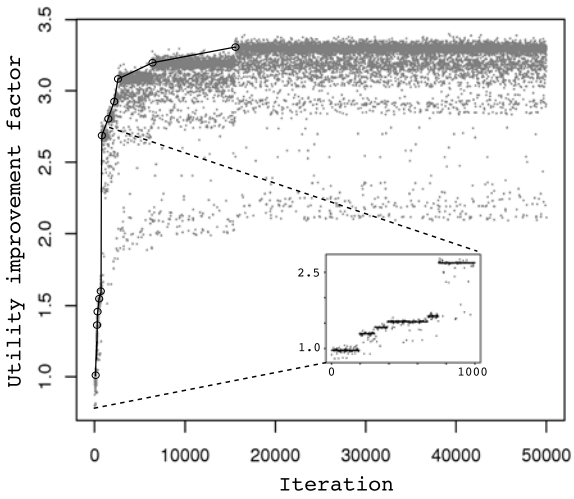

Figure 9: Improvements obtained by doing 50000 iterations of the 2-exchange operator for the EDF schedule generated during the 3000^{th} scheduling instance. The DEC data size distribution is used here. A newly obtained solution is considered better only if it exceeds the utility of the current solution by at least 20.

generate a new one. If the utility improves by more than 20, the new schedule 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 factor of improvement is computed as *utility of solution/utility of EDF solution*. A gray point (x, y) in the plot illustrate that the factor of improvement for the solution obtained in the x^{th} iteration of the search is y. The solid line joins the points where a generated schedule had an utility improvement of 20, 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 operator) have, more or less, equal amounts of utility associated with them. Hence the improvement factor values accumulate around a region to generate the bands. A more interesting observation is the amount of improvement obtained across the different iterations. We see that the schedule utility improves to a factor of 2.5 within the first 1000 iterations of the local search (inset Fig. 9), 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 much better schedule (observe the long horizontal band stretching from around the 15000 to the 50000 iteration). Hence, local search mechanisms are only useful when the initial schedules generated by the heuristics are not "good" ones.

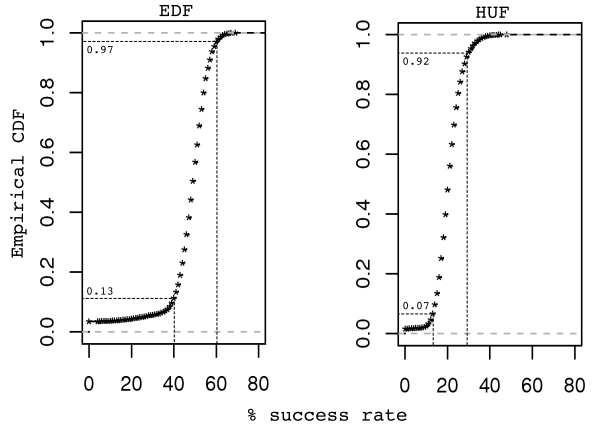

Figure 10: Empirical cumulative density function of the % success rates of performing 100 independent iterations of the 2-exchange operation in each scheduling instance with EDF (left) and HUF (right). A success means the operation resulted in a better schedule.

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. 10. 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.

The observations till this point leaves us with 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. Thereby, combining local search to both the heuristics enable us to at least "climb up" the easily reachable points in the search space.

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

Our justification as to why local search with an operator like 2-exchange fails after a certain extent is based on the fact that these operators are limited by the number of points they can sample – the neighborhood. This can also be verified from Fig. 9, where the appearance of thin bands are indicative of the low sampling of the area. 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 [4, 12]. This is the primary motivation behind using a recombinative ES to get improved schedules.

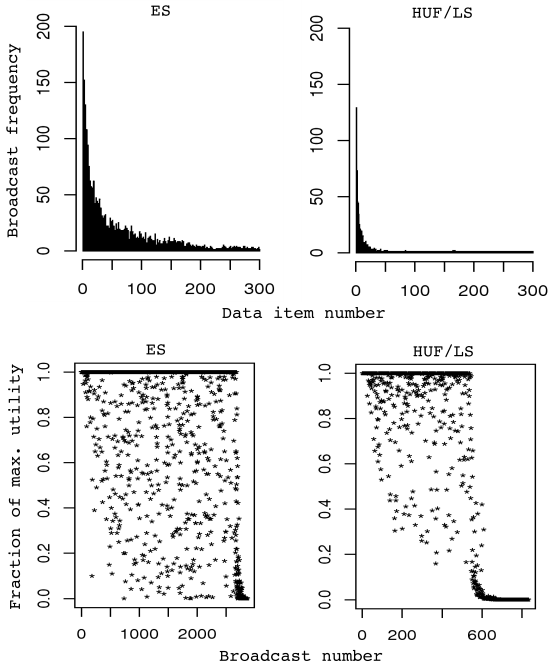

Figure 11: Top: Broadcast frequency of the N data items for ES (left) and HUF/LS (right). Bottom: Fraction of maximum utility obtained from the different broadcasts for ES (left) and HUF/LS (right). The DEC distribution is used with a 80 KB/s bandwidth.

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. 11 (top) shows the frequency distribution. A clear distinction is observed in the frequencies for average 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 which 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 an uniform manner. As can be seen from Fig. 11 (bottom), the pitfalls of the absence of this balance in HUF/LS is observed after a majority of the broadcasts have 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.

This insight suggest that heuristics that can take into consideration the expected broadcast frequency of data items and their relative sizes should do well in the context of data utility. Probabilistic [26] and disk-based [1] broadcasts employ these notion for push based architectures. It is thus worth investigating how they can be tailored for on-demand architectures. Further, balancing the utility losses and gains is a crucial aspect that any well performing heuristic needs to take into consideration.

6.4 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 a scheduling decision. 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.

7. CONCLUSIONS

In this paper, we address the problem of time critical data access in pervasive 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 the experiments. Our initial observation on their performance conform to the speculation that HUF performs better since it takes into consideration the utility of requests while making scheduling decisions. Further analysis of the nature of the scheduling problem show that EDF and HUF generated schedules can be greatly improved by introducing a minor amount of local search to them. The impact of the local search is found to be a direct consequence of the schedules generated by these heuristics, which are found to belong to a region of the search space from where obtaining improvements is not difficult.

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, and often suboptimal solutions generated by such a technique can be better than one obtained from a heuristic.

Future work in this context is inspired from the insights obtained from the analysis conducted on the $(2 + 1)$-ES. From an utility standpoint, we intend to explore the option of designing heuristics that pay special attention to factors like broadcast frequency and loss-gain trade-off during scheduling decisions. Also, the current analysis can be extended to include other heuristics, and the case when broadcasts involve serving requests for ordered data items. When requests involve multiple data items which may undergo regular updates, the validity of a broadcast is to be taken into account. Timeliness delivery of a data item then has to consider a validity deadline as well.

8. 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.

9. REFERENCES

[1] Acharya, S., Alonso, R., Franklin, M., and Zdonik, S. Broadcast Disks: Data Management for Asymmetric Communication Environments. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data* (San Jose, CA, USA, 1995), pp. 199–210.

[2] Acharya, S., and Muthukrishnan, S. Scheduling On-Demand Broadcasts: New Metrics and Algorithms. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking* (Dallas, TX, USA, 1998), pp. 43–54.

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

[4] Beyer, H. An Alternative Explanation for the Manner in which Genetic Algorithms Opearate. *BioSystems 41* (1997), 1–15.

[5] Beyer, H., and Schwefel, H. Evolution Strategies: A Comprehensive Introduction. *Natural Computing 1* (2002), 3–52.

[6] Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S. Web Caching and Zipf-Like Distributions: Evidence and Implications. In *Proceedings of the IEEE INFOCOM '99* (New York, NY, USA, 1999), pp. 126–134.

[7] Buttazzo, G., Spuri, M., and Sensini, F. Value vs. Deadline Scheduling in Overload Conditions. In *Proceedings of the 16th IEEE Real-Time Systems Symposium* (Pisa, Italy, 1995), pp. 90–99.

[8] Cho, H., Wu, H., Ravindran, B., and Jensen, E. D. On Multiprocessor Utility Accrual Real-Time Scheduling With Statistical Timing Assurances. In *Proceedings of the IFIP International Conference on Embedded and Real-Time Ubiquitous Computing* (Seoul, Korea, 2006), pp. 274–286.

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

[10] Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

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

[12] Holland, J. *Hidden Order: How Adaptation Build Complexity*. Basic Books.

[13] Jensen, E., Locke, C., and Tokuda, H. A Time Driven Scheduling Model for Real-Time Operating Systems. In *Proceedings of the Sixth IEEE Real-Time Systems Symposium* (San Diego, CA, USA, 1985), pp. 112–122.

[14] Jiang, S., and Vaidya, N. H. Scheduling Data Broadcasts to "Impatient" Users. In *Proceedings of the First ACM International Workshop on Data Engineering for Wireless and Mobile Access* (Seattle, WA, USA, 1999), pp. 52–59.

[15] Kim, J.-H., and Chwa, K.-Y. Scheduling Broadcasts with Deadlines. *Theoretical Computer Science 325*, 3 (2004), 479–488.

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

[17] Lee, K. C. K., Lee, W.-C., and Madria, S. Pervasive Data Access in Wireless and Mobile Computing Environments. *Wireless Communications and Mobile Computing* (in press).

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

[19] Li, P. A Utility Accrual Scheduling Algorithm for Real-Time Activities with Mutual Exclusion Resource Constraints. *IEEE Transactions on Computers 55*, 4 (2006), 454–469.

[20] Page, A. J., and Naughton, T. J. Framework for Task Scheduling in Heterogeneous Distributed Computing Using Genetic Algorithms. *Artificial Intelligence Review 24*, 3-4 (2005), 415–429.

[21] Ravindran, B., Jensen, E. D., and Li, P. On Recent Advances in Time/Utility Function Real-Time Scheduling and Resource Management. In *Proceedings of the Eight IEEE International Symposium on Object-Oriented Real-Time Distributed Computing* (Seattle, WA, USA, 2005), pp. 55–60.

[22] Rechenberg, I. *Evolutionsstrategie: Optimierung technischer Systemenach Prinzipien der biologischen Evolution*. PhD thesis, Technical University of Berlin, 1970.

[23] Starkweather, T., McDaniel, S., Whitley, C., Mathias, K., and Whitley, D. A Comparison of Genetic Sequencing Operators. In *Proceedings of the Fourth International Conference on Genetic Algorithms* (San Diego, CA, USA, 1991), pp. 69–76.

[24] Su, C.-J., and Tassiulas, L. Broadcast Scheduling for Information Distribution. In *Proceedings of the INFOCOM '97* (Kobe, Japan, 1997), pp. 109–117.

[25] Vengerov, D., Mastroleon, L., Murphy, D., and Bambos, N. Adaptive Data-Aware Utility-Based Scheduling in Resource-Constrained Systems. Tech. Rep. TR-2007-164, Sun Labs, 2007.

[26] Wong, J. W. Broadcast Delivery. *Proceedings of the IEEE 76*, 12 (1988), 1566–1577.

[27] Wu, H., Balli, U., Ravindran, B., and Jensen, E. D. 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* (Hong Kong, 2005), pp. 213–219.

[28] Wu, X., and Lee, V. C. Wireless Real-Time On-Demand Data Broadcast Scheduling with Dual Deadlines. *Journal of Parallel and Distributed Computing 65*, 6 (2005), 714–728.

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

[30] Xuan, P., Sen, S., Gonzalez, O., Fernandez, J., and Ramamritham, K. Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments. In *Proceedings of the Third IEEE Real-Time Technology and Applications Symposium* (Montreal, Canada, 1997), pp. 38–48.