Optimizing Real-Time Ordered-Data Broadcasts in Pervasive Environments Using Evolution Strategy

Rinku Dewri, Darrell Whitley, Indrajit Ray and Indrakshi Ray

Colorado State University, Fort Collins, CO 80523, USA
{rinku,whitley,indrajit,iray}@cs.colostate.edu

Abstract. We consider the problem of real-time data broadcast scheduling in pervasive systems with soft deadlines and constraints on the order in which data items should be broadcast to be useful. The broadcast schedule needs to be generated to provide a certain level of quality of service. Thus, the real-time scheduler has to effectively trade-off between its running time and the quality of schedules generated. We use an evolution strategy to solve the problem. The variants tested includes $(1+\lambda)$-ES, $(1, \lambda)$-ES, and a $(2+1)$-ES with a modified Syswerda recombination operator, as well as a genetic algorithm.

Keywords: Data broadcasting, Scheduling, Evolution strategy

1 Introduction

Many pervasive application domains involve clients interested in groups of related data items that can be processed one at a time following some order. Consider the following example – a traffic management system in a large city. The system gathers current traffic information using sensors and disseminates it to drivers in real time on demand. Drivers use smart GPS navigation units to request road conditions so that they can be routed and re-routed along the best possible roads. The GPS unit periodically requests traffic information for the roads that are still remaining in its route plan. The traffic service needs to provide the road conditions in the same order that the roads are in the route plan. That is, if the current route plan is $\langle r_1, r_2, r_3, r_4 \rangle$, where r_i is a road identifier, then the traffic service should provide the information as $\langle rc_1, rc_2, rc_3, rc_4 \rangle$, where rc_i is the road condition for r_i. A scheduling problem occurs in such an application when the number of data access requests is larger than the bandwidth capacity of the server.

Various soft deadlines may also be imposed on the requested data items, which if not served within a specific time window may result in the data item having near zero utility when finally received. For example, from the time that a driver makes a request for traffic information on road r_i to the time the monitoring service reports back with a gridlock on r_i, the driver may already have

missed a more convenient exit for an alternate route. Given the resource limitations, it is not always possible that the time constraints of every incoming request be satisfied. Thus, a broadcast schedule is sought which can serve clients with as much utility as possible. The scheduling problem is more difficult in a real-time setting where generation of a high utility schedule has to respect run time constraints as well.

In this paper, we first propose an utility accrual method for data requests involving constraints on the order of the requested data items. Second, we define a modified version of the Syswerda recombination operator for use in methods with recombination. Finally, the utility function is used by an evolution strategy based schedule optimizer to evaluate the effectiveness of the schedules. We pay particular attention to the run time constraint of the scheduler and argue that simple variants of evolution strategy can be employed to satisfy this requirement.

The remainder of the paper is organized as follows. Section 2 outlines the related work. Section 3 presents the utility function and a formal statement of the problem. Section 4 presents the specifics of the evolution strategy method. The experimental setup is presented in Sect. 5. Discussion on the results obtained are presented in Sect. 6. Finally, Sect. 7 concludes the paper.

2   Related Work

Several shortcomings of using a strict deadline based system are discussed by Ravindran et al. [1]. 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 [2]. They introduce the idea of $Time - Utility\ Functions$ to capture the semantics of soft time constraints specifying utility as a function of completion time. An attempt to understand the benefit of utility values in hard deadline scheduling algorithms is made by Buttazzo et al. [3]. Wu et al. study a task scheduling problem where utility is considered a function of the start time of the task [4].

Ordered queries have been studied in the mobile computing paradigm. Chehadeh et al. take into consideration object graphs to cluster related objects close to one another in the broadcast channel [5]. Hurson et al. propose an extension for multiple broadcast channels [6]. 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 [7].

Our approach to the problem differs in the introduction of an utility metric to evaluate the goodness of schedules and uses evolution strategy as a fast and effective method to obtain high utility schedules.

3   Broadcast Scheduling

Data broadcasting is an efficient approach to address data requests, particularly when similar requests are received from a large user community. *Push-based* architectures broadcast commonly accessed data at regular intervals. *On-demand*

architectures allow the clients to send their requests to the server. Certain requests have an added requirement of data items being served in a particular order. Further, particular emphasis has to be paid to the time criticality and utility of a served request in an on-demand architecture.

3.1 Broadcast Model

Clients use an uplink channel to a data provider to request various data items served by the provider. Each request Q_j takes the form of a tuple $\langle D_j, R_j \rangle$, where R_j is the response time within which the requesting client expects the first data item from the ordered set D_j, hereafter called a *data group*. The assumption that processing at the client end can start as soon as the first data item in the group is received is implicit in this context. Nonetheless, a client must receive all requested data items for the processing to complete. 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 dynamic in nature. A request is considered to be *"fully served"* when the last data item in the requested data group is retrieved, otherwise it is considered *"partially served"*.

The scheduler is invoked every time a new request is received. At each instance, the scheduler first determines the requests that will be fully served by the current broadcast and removes them from the request queue. For the remaining requests, the data items required to serve them are determined and a schedule is generated. The scheduler makes a "best effort" at generating a schedule that respects the response time requirements of the clients.

3.2 Utility Metric

In order to facilitate soft deadlines, we make the assumption that the utility of a data item received by a client decreases exponentially if not received within the expected response time. For request Q_j arriving at time T_j and involving the data group $D_j = \{d_{1j}, d_{2j}, \ldots, d_{N_j j}\}$, let $t_{1j}, t_{2j}, \ldots, t_{N_j j}$ be the time when the respective data items are retrieved by the client. The utility generated by serving the first data item is given as,

$$u_j[t_{1j}] = \begin{cases} 1 & , t_{1j} - T_j \leq R_j \\ e^{-\alpha(t_{1j} - T_j - R_j)} & , t_{1j} - T_j > R_j \end{cases} \tag{1}$$

The utility from the subsequent items is then given as, for $i = 2, \ldots, N_j$ and inter-item response time R_T,

$$u_j[t_{ij}] = \begin{cases} u_j[t_{(i-1)j}] & , t_{ij} - t_{(i-1)j} \leq R_T \\ u_j[t_{(i-1)j}]e^{-\alpha(t_{ij} - t_{(i-1)j} - R_T)} & , t_{ij} - t_{(i-1)j} > R_T \end{cases} \tag{2}$$

The utility of a data item for a client decays by half as a function of the response time, i.e. $\alpha = \ln 0.5/R$, where $R = R_j$ for the first data item in the requested group and $R = R_T$ for any subsequent data item.

If all data items are broadcast in a timely manner, a maximum utility of 1 will be generated by each data item. However, when a data item's broadcast time exceeds its expected response time, not only will its utility drop, it will also influence the maximum utility that can be obtained from subsequent items. We then say that the utility generated by serving the request is given by the utility generated at the last item of the data group, or $U_j = u_j[t_{N_j j}]$. The quality of service desired in an application domain can be directly specified as a fraction of the utility derived from serving the requests. For a schedule S, generated to serve the requests $Q_1, Q_2, \ldots, Q_M$ in the queue, the utility generated by the schedule is the aggregation of the utilities for the requests in the queue, given as,

$$U_S = \sum_{k=1}^{M} U_k \qquad (3)$$

3.3 Problem Statement

A data source $D = \{D_1, D_2, \ldots, D_N\}$ is a set of N ordered sets (or data groups), where $D_j = \{d_{1j}, d_{2j}, \ldots, d_{N_j j}\}$ with N_j being the cardinality of D_j and $j = 1, \ldots, N$. All data items d_{ij} are assumed to be unique and are of equal size d_{size}. A request queue at any instance is a dynamic queue Q with entries Q_j of the form $\langle D_j, R_j \rangle$, $D_j \in D$ and $R_j \geq 0$. At an instance t_{curr}, let $Q_1, Q_2, \ldots, Q_M$ be the entries in Q. We define the notation $Wait[Q_j]$ to denote the data item that the request Q_j is currently waiting for. Further, we define $Rem[Q_j]$ as the ordered subset of data items that has been requested in Q_j but not yet received, i.e. $Rem[Q_j] \subseteq D_j$. A schedule is a total ordering of the elements in the multi-set $\bigcup_{j=1,\ldots,M} Rem[Q_j]$.

The time instance at which a particular data item from the schedule starts to be broadcast is dependent on the bandwidth of the broadcast channel. A broadcast channel of bandwidth b can transmit b *data unit* 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 $d_{i_1 j_1} < d_{i_2 j_2} < \ldots < d_{i_P j_P}$, the data item $d_{i_k j_k}$ can be retrieved by an interested client at time instance $t_{i_k j_k} = t_{ready} + [(k-1)d_{size}/b]$. All requests Q_j in Q with $Wait[Q_j] = d_{i_k j_k}$ are then partially served, i.e. $t_{i_k j}$ for such requests is set to $t_{i_k j_k}$, and $Rem[Q_j]$ is changed to $Rem[Q_j] - \{d_{i_k j_k}\}$. The request is fully served when $Rem[Q_j] = \phi$.

At each scheduling instance, the problem then is to find a schedule with maximum utility given by (3).

4 Solution Methodology

Evolution Strategies (ES) [8] are typically expressed by the μ and λ parameters. 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. We restrict our focus to simple stochastic local search variants by setting $\mu = 1$. We also experiment with a $(2 + 1)$-ES with recombination and a simple genetic algorithm (GA).

A schedule can be represented by a permutation of the data items currently in the $Rem[\cdot]$ sets of requests. Since the same data item may be present multiple times in this permutation, a request identifier is attached to every data item. For the requests $Q_1, Q_2, \ldots, Q_M$ currently in the request queue, the data items in $Rem[Q_j]$ of a request are identified by the tuples $\langle j, d_{i_k j} \rangle$, where $d_{i_k j} \in Rem[Q_j]$. The first component of a tuple identifies the request for which it exists in the permutation, and the second component identifies the data item number. The request identifier is only used in the permutation and is not part of the broadcast for the data item. Hence, if a request is waiting on a data item, say $d_{i_{k_1} j_1}$, then upon broadcast it will be retrieved by the request irrespective of the request identifier attached to the data item in the tuple, j_1 in this case.

The simplest ES methods employ a mutation operator only, which implies a low overhead on the running time of the scheduler in a dynamic setting. "Shift" mutation selects two random positions in a permutation and the element at the first chosen position is removed and inserted at the second chosen position. The second random position is chosen in a way such that the ordering constraints for the data item in the tuple is preserved.

Recombination for the (2+1)-ES and the GA is achieved by a modified version of the Syswerda order-based crossover operator [9]. Syswerda's operator chooses random positions on a parent for exchange with the other. However, doing so can disrupt the ordering constraints of the permutation on the offspring (Fig. 1). The modified operator, called the *constrained-Syswerda* operator, eliminates this problem by restricting the choice of positions to a random contiguous block instead. One can prove this modification always yields feasible schedules.

```
Valid Parent 1      :  a b c d e f g h i j
Valid Parent 2      :  a d e b c h i f g j
Cross Positions     :  *   *   *
Invalid Offspring   :  a b e d c f g h i j
```

Fig. 1. A counter example for the Syswerda order-based crossover operator. Ordering constraints are $\{a, b, c\}$, $\{d, e, f, g\}$ and $\{h, i, j\}$; constraint $\{d, e, f, g\}$ is violated in the offspring.

5 Experimental Setup

Due to the non-availability of a standard test dataset in the domain, the data used in our experiments is generated using well known distributions that are known to capture the dynamics of a public data access system [10]. The dataset

contains 10000 requests generated using a Poisson distribution with an arrival rate of 3 requests per second. Each request consists of an arrival time, data group number, and an expected response time for the first data item in the group. We consider 100 different data groups, the number of data items in each drawn from an exponential distribution with rate parameter 10. The bandwidth for the broadcast channel is set at $200KB/s$.

The data groups requested are assumed to follow the commonly used *Zipf* distribution [10] with the characterizing exponent of 0.8. Under this assumption, the first data group becomes the most requested one, while the last one is the least requested. Broadcast schedules can be heavily affected by the size of the most requested data group. Thus, we consider two different assignments: INC − most requested data group has the smallest number of data items, and DEC − most requested data group is the largest in size [11, 12]. Each data item is of size $50KB$.

Expected response times for the first data item are assigned from a normal distribution with mean 60s and standard deviation 20s. 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. Any negative value is replaced by zero. Response times for intermediate data items is set at 1s.

For different variants of the $(1 + \lambda)$-ES and $(1, \lambda)$-ES, we fixed the maximum number of function evaluations to 15000. The run time of a scheduling instance is around 0.01s (on a 2×2.66 GHz Xeon running Fedora Core 8 with 2GB memory) with this setting. The number of function evaluations is fixed at 5000 for the $(2 + 1)$-ES. The parameters for the GA is set as follows: population size = 100, number of function evaluations = 15000, crossover probability = 0.8, mutation probability = 0.05, and 2-tournament selection.

6 Empirical Results

We present the results obtained from different variants of the ES on the two different data group assignments − INC and DEC. The results are averaged for 20 runs for each variant. Figure 2 shows the percentage global utility obtained by running $(1 + \lambda)$-ES with different λ values. Recall that each request can have a maximum utility of 1. The percentage global utility is thus computed from the fraction of this maximum utility generated in the requests in the data set. For the INC type assignment, a $(1 + 1)$-ES yields an acceptably high ($> 90\%$) utility level. Given the bandwidth limit of $200KB/s$, up to 4 data items can be transmitted in a single time unit. In the case of an INC type assignment, this can serve at least 4 different requests for the frequently requested data groups. Note that the INC assignment has the most requested data group as the smallest in size, which is one data item in our experimental data set.

The DEC type assignment has 20 data items in the most frequently requested data group. Further, the *Zipf* distribution makes the larger data groups more
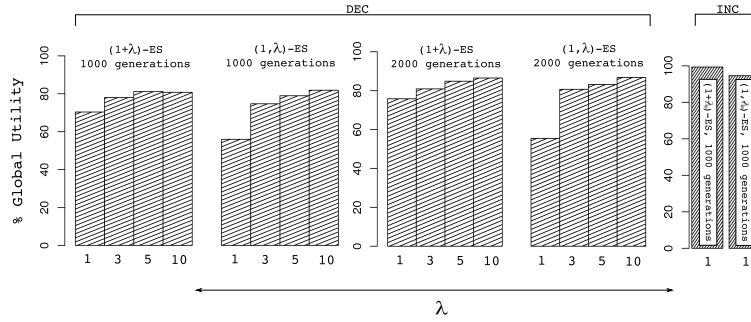
Fig. 2. Percentage global utility for the different λ values in the ES. For the *INC* assignment, a high utility level is obtainable with $\lambda = 1$. For the *DEC* assignment, increasing the sampling rate λ and the number of generations results in improvement of the global utility.

often requested than the smaller ones. The $200KB/s$ bandwidth poses a hard bottleneck in this situation. Quite often, different requests for the same data group arrive at different times prohibiting the $Wait[\cdot]$ value of those requests to be the same. The $(1 + 1)$-ES fails to provide the same level of performance as it does for the *INC* assignment. Increasing the sampling rate λ to $3, 5$ and 10 show improvements up to 80% utility levels. Although increasing the number of generations to 2000 improved the utility level up to 86%, the number of function evaluations ($2000 \times 10 = 20000$) exceeded the maximum set limit of 15000.
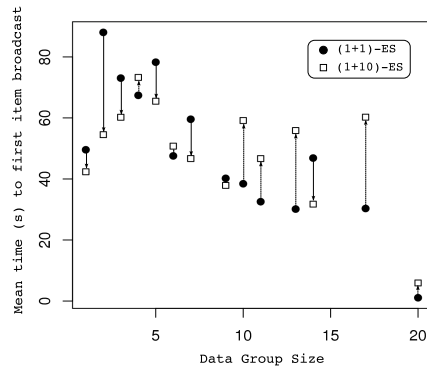

Fig. 3. Mean difference between time of request arrival and time of first data item broadcast for data groups of different sizes in the *DEC* assignment.

The primary difference between the schedule utilities generated by $(1 + 1)$-ES for 1000 generations and $(1 + 10)$-ES for 2000 generations is attributable to the latency that the scheduler puts in between the time of arrival of a request and the time when the first data item for the request is broadcast. Figure 3

shows the mean values of this latency for data groups of different sizes. The $(1+10)$-ES maintains a higher mean latency for the larger data groups, whereas the $(1+1)$-ES maintains a higher value for the smaller data groups. Given that most requests are for the larger data groups in the DEC assignment, postponing the first data item broadcast for such requests provides gaps in the schedule to serve pending (or partially served) requests. Recall that the expected response time is the highest for the first data item (between 0 and 120s). Once a client has received the first data item, the expected response time drops to $R_T = 1s$. Hence, by delaying the first data item broadcast for the most requested data groups, the $(1+10)$-ES maintains a better flexibility in serving pending requests.

For the experimental data set, a simple $(1+1)$-ES for 1000 generations is sufficient when the QoS requirement is not too high ($< 75\%$). For the case when the utility requirement is higher, a higher sampling rate is desired. However, note that results obtained from running the different variants for 2000 generations (more function evaluations) do not always yield a high difference in the utility levels as compared to those from running the same variants for 1000 generations. This is observed in both the *comma* and *plus* variants of the ES.
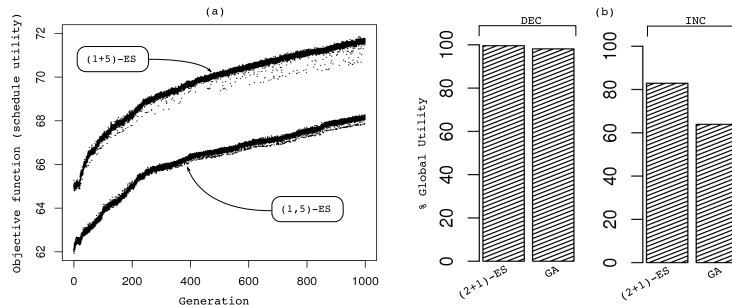

Fig. 4. (a) Schedule utility progress during the 5000th scheduling instance with DEC. Utilities are also plotted for 100 points sampled around the parent of every generation. (b) Percentage global utility for $(2+1)$-ES and a genetic algorithm.

Figure 4a shows the changes in the objective function (schedule utility given by (3)) value during the scheduling instance when the 5000th request arrives. The scheduler is working with DEC and $\lambda = 5$. Further, at each iteration, the plot shows the utilities of 100 randomly sampled points (using shift mutation) near the current parent. Note that the rise in the utility of the schedule is faster during the first 250 generations and then slows down considerably. In other words, as better schedules are obtained, improvements are harder to find. This in turn makes the progress slower. Moreover, the randomly sampled points around the parent of the current generation show very small differences in the utility values. This makes it difficult for the ES to maintain a steady increase in the schedule utility.

Figure 4b shows the percentage global utility generated by the $(2+1)$-ES and the GA. Performance of both methods is on a par with the stochastic search variants for the INC size distribution. The GA's performance on the DEC distribution is easily overpowered by a $(1+3)$-ES. Further, the GA does a maximal utilization of the allowed function evaluations of 15000. The $(2+1)$-ES achieves an utility of 83%, equivalent to that of the $(1+5)$-ES with 2000 generations. An important factor to consider here is the number of function evaluations used in the two methods − 5000 in $(2+1)$-ES compared to 10000 in the $(1+5)$-ES. Although this performance is marginally better (about 3%) than the $(1+5)$-ES with 1000 generations, we stress that even obtaining such marginal improvements is difficult with the DEC distribution.

The enhanced performance of $(2+1)$-ES is attributable to the additional exploration brought forth by the recombination operator. Local search methods do not display enough exploratory capabilities when stuck in a plateau of the search space. Recombination allows for a more diverse sampling in such cases, thereby resulting in a faster exploration through plateaus. The GA is expected to benefit from this as well. The cause of its poor performance is not well understood.

Scheduling time is an important factor to consider in any dynamic scheduler. However, it should be noted that the amount of time a scheduler has to generate a schedule need not be always fixed. In this study, the scheduler is triggered only when a new request arrives. The scheduler otherwise remains idle, including the time when a broadcast is ongoing. In a real scenario, it may be beneficial to trigger the scheduler more often − for example when a new broadcast starts − and allow for more exploration of the search space.

7   Conclusions

Pervasive computing applications often need to broadcast grouped data objects such that the elements in the group satisfy a user specified ordering constraint. In addition, objects not served within a specific window may end up having near zero utility. We introduce a method of utility accrual for grouped data objects and use it to evaluate the effectiveness of a schedule. We argue that evolution strategy is a viable methodology to maximize the utility of broadcasts given the run time constraints of the application. We investigate three different methods − a simple stochastic local search using $(1+\lambda)$-ES and $(1,\lambda)$-ES, a $(2+1)$-ES with a modified Syswerda recombination operator, and a genetic algorithm. Our experiments suggest that the generation of an optimal schedule when the most requested group has the highest number of data items is a difficult problem to solve, often requiring a longer duration of search. However, recombination based ES appears to be particularly effective. The $(2+1)$-ES with the proposed recombination operator demonstrates the potential to generate better schedules without engaging in too many function evaluations, thereby providing a fair trade-off between the run time and utility objectives of a scheduler.

The problem considered in this paper assumes that data items are unique across different data groups. However, there exists other problems in pervasive

environments where the data groups can have common data items. In future, we plan to investigate if the scheduling strategy identified here works equally well in such problem domains.

Acknowledgment

References

1. Ravindran, B., Jensen, E.D., Li, P.: On Recent Advances in Time/Utility Function Real-Time Scheduling and Resource Management. In: 8th IEEE ISORC. (2005) 55–60
2. Jensen, E., Locke, C., Tokuda, H.: A Time Driven Scheduling Model for Real-Time Operating Systems. In: 6th IEEE RTSS. (1985) 112–122
3. Buttazzo, G., Spuri, M., Sensini, F.: Value vs. Deadline Scheduling in Overload Conditions. In: 16th IEEE RTSS. (1995) 90–99
4. Wu, H., Balli, U., Ravindran, B., Jensen, E.D.: Utility Accrual Real-Time Scheduling Under Variable Cost Functions. In: 11th IEEE RTCSA. (2005) 213–219
5. Chehadeh, Y., Hurson, A., Kavehrad, M.: Object Organization on a Single Broadcast Channel in the Mobile Computing Environment. Multimedia Tools and Applications 9(1) (1999) 69–94
6. Hurson, A., Chehadeh, Y., Hannan, J.: Object Organization on Parallel Broadcast Channels in a Global Information Sharing Environment. In: 19th IEEE IPCCC. (2000) 347–353
7. Huang, J.L., Chen, M.S.: Dependent Data Broadcasting for Unordered Queries in a Multiple Channel Mobile Environment. IEEE Transactions on Knowledge and Data Engineering 16(9) (2004) 1143–1156
8. Rechenberg, I.: Evolutionsstrategie: Optimierung technischer Systemenach Prinzipien der biologischen Evolution. PhD thesis, Technical University of Berlin (1970)
9. Syswerda, G.: Schedule Optimization Using Genetic Algorithms. In Davis, L., ed.: The Genetic Algorithms Handbook. (1990)
10. Breslau, L., Cao, P., Fan, L., Phillips, G., Shenker, S.: Web Caching and Zipf-Like Distributions: Evidence and Implications. In: IEEE INFOCOM. (1999) 126–134
11. Hameed, S., Vaidya, N.: Efficient Algorithms for Scheduling Data Broadcast. Wireless Networks 5(3) (1999) 183–193
12. Lee, V.C., Wu, X., Ng, J.K.Y.: Scheduling Real-Time Requests in On-Demand Data Broadcast Environments. Real-Time Systems 34(2) (2006) 83–99