Noname manuscript No.
(will be inserted by the editor)

Real Time Stochastic Scheduling in Broadcast Systems with Decentralized Data Storage

the date of receipt and acceptance should be inserted later

Abstract Data broadcasting is an efficient method to disseminate information to a large group of requesters with common interests. Performing such broadcasts typically involve the determination of a broadcast schedule intended to maximize the quality of service provided by the broadcast system. Earlier studies have proposed solutions to this problem in the form of heuristics and local search techniques designed to achieve minimal deadline misses or maximal utility. An often ignored factor in these studies is the possibility of the data items being not available locally, but rather have to be fetched from data servers distributed over a network, thereby inducing a certain level of stochasticity in the actual time required to serve a data item. This stochasticity is introduced on behalf of the data servers which themselves undergo a dynamic management of serving data requests. In this paper we revisit the problem of real time data broadcasting under such a scenario. We investigate the efficiency of heuristics that embed the stochastic nature of the problem in their design and compare their performance with those proposed for non-stochastic broadcast scheduling. Further, we extend our analysis to understand the various factors in the problem structure that influence these heuristics, and are often exploited by a better performing one.

Keywords Data broadcasting, Stochastic scheduling

Second Revision.

1 Introduction

Many application domains involve groups of clients interested in the same data item. Broadcasting serves as an efficient method of disseminating data in such settings since multiple clients can be served with a single broadcast. *Push-based* architectures broadcast commonly accessed data at regular intervals without clients explicitly requesting them. 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 items, serves multiple clients at a time. Various *hard* or *soft* deadlines may 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. 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 as many clients as possible within their deadlines. The scheduling problem is more difficult in a real-time setting where generation of a schedule has to respect run time constraints as well.

Most of the earlier works in designing broadcast schedulers assume a centralized system where the broadcast server has local access to the data items (Acharya and Muthukrishnan, 1998; Aksoy and Franklin, 1999; Dewri et al., 2008; Lee et al., 2006; Xu et al., 2006). This simplifies the problem since the broadcast scheduler need not take into consideration the time required to retrieve the data item while making scheduling decisions. However, a number of application domains exist where such centralized storage of data are not possible. Consider the example of a company that provides investment consulting and management services to its clients (a company like Morningstar®). One service provided by such a company is real-time, on-demand information on stocks, mutual funds etc., in the form of market indices. Typically, such a company does not generate this information itself but fetches it from other companies (Morningstar®, for example, gets a significant portion of this data from the company Interactive Data Corporation℠). With the current ubiquity of wireless computing devices, we can imagine that clients of the company seek and receive periodic market data on their mobile phones, PDAs and notebooks, and analyze the data to determine which investments are rewarding. Since multiple clients may seek the same data, it makes good business sense for the company to broadcast this data. The clients also perform financial transactions using such devices. Market indices change throughout the day and it is important to analyze such trends along multiple dimensions in order to perform trading. Thus, although queries from clients are implicitly associated with deadlines, these can be considered soft. Even if the queries are served after their deadline, they still have some value. The overall goal of the company will be to satisfy as many requests as possible in a timely manner, keeping in mind that the data may need to be fetched from other sources.

Decentralized storage of data introduces a novel problem to the broadcast scheduler, specifically if the data items must be retrieved from a data server prior to broadcast. Typically, the scheduler has complete knowledge on the time required to broadcast an item and uses this information in deciding a schedule. However, when data have to be fetched from a data server, this knowledge assumes a stochastic form. A data server will usually be serving multiple broadcast servers following its own request management policy. Thus, the time required to fetch a data item is not known apriori. A broadcast scheduler then has to build a schedule based on stochastic information available about the retrieval time of data items.

In this paper, we visit the problem of data broadcast scheduling under such a scenario. We model the problem as a case of *stochastic scheduling*[1] and explore the performance of a number of heuristic based schedulers. We argue that heuristics used in deterministic scheduling may not perform well in a stochastic problem of this nature. We show how probability estimates on a data server's response times can be used in the design of better schedulers. To this end, we propose two heuristics – the *Minimum Deadline Meet Probability* and the *Maximum Bounded Slack Probability* heuristics – that are based on the request completion time distributions rather than their exact values. Further, we augment our observations with an analysis to understand the underlying principles of a good scheduler. Our results demonstrate that a better performing broadcast policy exploits factors such as bandwidth utilization, data sizes and access frequencies. This is often as a result of how the specific heuristic has been formulated.

The remainder of the paper is organized as follows. Section 2 discusses the related work in broadcast scheduling. Section 3 presents the broadcast architecture used in this study. The formal statement of the problem and the performance metrics used are also outlined here. The heuristics experimented with are discussed in Section 4. Section 5 introduces the two heuristics proposed in this paper for stochastic broadcast scheduling. Section 6 outlines the details of the experimental setup and the synthetic data set used to evaluate the performance of the heuristics. Section 7 presents the results obtained and provides an extensive discussion on the performance of the heuristics. Finally, Section 8 summarizes the paper with references to future work.

2 Related Work

Data broadcasting has received extensive attention in wireless communication systems. Su and Tassiulas present a dynamic optimization problem to generate schedules with minimum access latency (Su and Tassiulas, 1997). Acharya and Muthukrishnan introduce the *stretch* metric to fairly evaluate the performance of schedules when response times of requests can vary widely due to differences in the size of data items requested (Acharya and Muthukrishnan, 1998). They propose dynamic optimization problems that utilize the stretch values in determining schedules that result in minimum deviation of response times from

[1] The term stochastic scheduling has been used elsewhere in a different context to signify probabilistic access frequencies.

the service time of requests. Sun et al. propose the LDCF algorithm to account for various cost factors typically observed in broadcast systems (Sun et al., 2003). Factors such as access time, tuning time and cost of handling failure requests are used to compute a delay cost for data items, which then serves as a priority measure for the items. Aksoy and Franklin propose the RxW heuristic to balance the popularity and urgency of requests (Aksoy and Franklin, 1999). However, most of these heuristics do not consider the time critical nature of requests in their formulation.

A hybrid broadcasting approach is proposed by Fernandez and Ramamritham to cater to the deadline requirement (Fernandez and Ramamritham, 2004). Wu and Lee consider constraints such as request deadlines and temporal data validity while proposing the RDDS algorithm (Wu and Lee, 2005). The RDDS algorithm assigns priority levels to each requested data item and broadcasts them in order of their priorities. Similar priority assignments are performed in the PRDS algorithm by Lee et al. (Lee et al., 2006). Their priority assignment is based on the urgency of requests, access frequency and the size of data items. Xu et al. propose the SIN-α algorithm for equi-sized data items (Xu et al., 2006). The algorithm is based on the slack and urgency of requests.

Attempts are rare in broadcast scheduling where deadline requirements are explored from an utility standpoint. Ravindran et al. discuss several shortcomings of using hard deadlines in real time scheduling systems (Ravindran et al., 2005). *Time-utility* functions were first proposed by Jensen et al. to show how soft deadlines can be more useful in realizing a value based system (Jensen et al., 1985). A similar study by Buttazzo et al. also point at the same (Buttazzo et al., 1995). Dewri et al. propose the use of heuristics based on evolutionary algorithms to explicitly maximize the utility of requests in broadcast scheduling (Dewri et al., 2008). They also show how local search techniques can improve the performance of simple heuristics.

While considerable attention has been paid to designing heuristics for broadcast scheduling in centralized data systems, heuristics for non-local data models are rare to find. Data staging concerns in broadcast scheduling were first highlighted by Aksoy et al. (Aksoy et al., 2001). The authors focus on a decision problem where the data items are not readily available for broadcast. An *opportunistic scheduling* mechanism is introduced where data items available for broadcast are chosen instead. Further, *hint-based cache management* and *prefetching* techniques are proposed in order to decrease the fetch latency of a data item. However, these techniques do not utilize any information available on the fetch time of data items while determining the broadcast schedule. The focus in our work is to supplement such data staging mechanisms by informing the scheduler on the workload present on a data server and make decisions accordingly.

Traintafillou et al. study a similar problem in the context of disk access latencies (Triantafillou et al., 2002). The primary objective of their work is to study the interplay among different components of a broadcast system, namely the broadcast scheduler, disk scheduler, cache manager and the transmission scheduler. However, disk access latencies are typically much smaller than data retrieval latencies from a secondary source. In effect, the impact of data fetch times is amplified in decentralized storage systems. Further, unlike as in non-local data systems, the heuristics proposed in their work assume that disk service times are known apriori.

Hierarchical data dissemination is used by Omotayo et al. in the context where data updates are frequently pushed from one server to another (Omotayo et al., 2006). The scheduling problem is explored at the *primary* server which needs to decide how frequently to broadcast updated data items to *secondary* servers. Since the primary server does not maintain any incoming request queue, there is no stochasticity involved on part of the secondary servers. Notably, stochastic scheduling has been explored in depth for job shop scheduling problems where execution time of jobs are not known apriori (Dempster et al., 1982; Megow et al., 2006).

3 Problem Modeling

Stochastic scheduling refers to the class of scheduling problems where the time required to complete a particular task is modeled as a random variable. Compared to deterministic scheduling where the time of completion of a task can be computed before it begins execution, the actual completion time in the stochastic case is known only after the task finishes execution. However, a probability distribution of the completion time is known (or can be computed) for such tasks. Data broadcast scheduling transforms into a stochastic problem when the data items to be broadcast are not available locally at the *broadcast server*. In such a scenario, the broadcast server has to retrieve the data items prior to initiating the broadcast, from *data servers* distributed over a network. The data servers on the other hand will be receiving multiple requests from other broadcast servers and will have their own policy to manage the serving of accumulating data requests. The serving of requests on part of the data server may be accomplished by a unicast, multicast or a broadcast mechanism. Hence, the time required by a broadcast server to fetch a data item will not be known until the item is actually fetched. Scheduling decisions on the broadcast server that must be made prior to fetching any data item will therefore be inaccurate. As an alternative, a broadcast server can use *response time* probability estimates of the data servers. The response time is the time elapsed between making a request and getting the request served. The probability estimates would present a broadcast server with likely durations of time it would require to retrieve a data item, thereby introducing the *stochastic broadcast scheduling* problem. A stochastic scheduler can use the probability estimates as prior knowledge on the problem instance, but does not utilize anticipated information from the future (such as a possible value for the response time) while generating a scheduling policy. Often times, the policy itself may undergo revisions as data items are actually fetched and their response times are known.

3.1 Broadcast Server Model

Fig. 1 depicts the broadcast architecture used in this study. Clients request different data items from a broadcast server using the uplink channel. Each request Q has an arrival time, a data item index and an absolute deadline, given by the tuple $\langle arr_Q, d_Q, dln_Q \rangle$. Although deadlines can be explicitly specified by the client, it is not a strict restriction. For example, the broadcast server can assign deadlines to the scheduler based on the current status (say a geographic location) of the client making the request. A single request involves only one data item. If multiple data items are required by a client, then multiple requests are formulated to retrieve them, with no constraint on the order in which they are to be served.

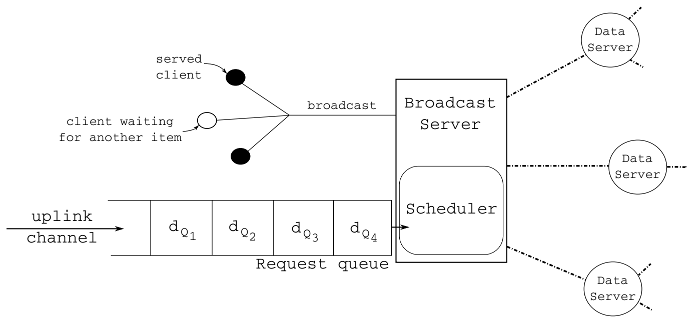

Fig. 1 Broadcast server architecture.

Data items are distributed across multiple data servers. The broadcast server has fast dedicated connections to these servers. The data servers have their own policies to serve incoming requests. The only knowledge the broadcast server has about the data servers are the data items they host and a probability mass function (pmf) of the response times of these servers. The pmfs can either be sampled from a known probability distribution function, or constructed by observing the response times of the data servers over a period of time.

The broadcast server reads the requests in the queue and invokes a scheduler to determine the order in which the data items are to be broadcast. Once a schedule is built, the broadcast server requests the corresponding data item from the respective data server and stores it in a buffer. For the sake of simplicity, we do not assume a system where the fetching of a data item and its broadcasting happens in parallel. Such an approach can raise synchronization issues which then have to be handled with multiple levels of buffering. Pre-fetching of data items (retrieving multiple items and holding in local storage) is also discarded in this model since the scheduler may decide to change the schedule to accommodate more urgent requests, in which case pre-fetched data items may have to be discarded. Also, the application domains we consider are where data items may undergo frequent updates at the data server. Nonetheless, all these factors can be accommodated in the broadcast server for added complexity. The broadcast server initiates the broadcast for the data item in a downlink channel of bandwidth b_{bs} as soon as the entire data item has been received from the data server. Interested clients retrieve the data item from the downlink channel. A request is served at the time instance, called the *completion time* of the request, when the entire data item is received by the client.

The scheduler is invoked every clock tick. During periods when the broadcast server does not receive any new request, the scheduler revisits the current schedule at specified time intervals. This is done since the scheduler gains precise information on the response times of the requests with each completed broadcast and may decide to change the schedule (generated with stochastic information) over time.

3.2 Data Server Model

A data server hosts a number of data items and maintains dedicated connections to multiple broadcast servers. It is also possible that the data server does not locally host the requested data item, but just acts as a designated channel to retrieve the item. In such a case, the data server has to fetch the item from another server. This facet can introduce multiple layers in the data fetching scheme. However, in our model, such multi-layered fetching schemes remain transparent to the broadcast server. This is because the broadcast server can only acquaint itself with response time pmfs about the data server. When such a scheme is in place, the response time pmfs will appropriately reflect the delay introduced. In this study, we assume that data items are locally hosted at the data servers.

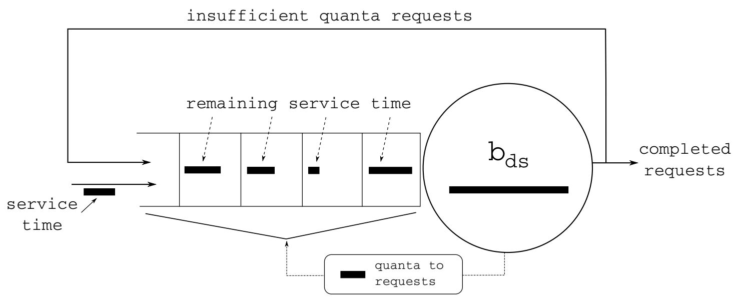

Fig. 2 Data server architecture following a $M/G/1$-PS model.

A data server is modeled similar to a $M/G/1$ queuing system with the processor sharing (PS) discipline. The $M/G/1$-PS queuing model assumes that requests arrive at a data server following an exponential distribution (Poisson in this case) and has arbitrary *service times*. The service time of a request is the time required to complete the request had it been the only one in the system. Completion of a request in this case implies transmission of the entire data item from the data server to the broadcast server. The bandwidth available at the data server is distributed uniformly to serve requests in a round-robin fashion. Hence, if there are n requests ready to be served in the queue and b_{ds} is the bandwidth available at the data server, then each request receives a quanta $\frac{b_{ds}}{n}$ of the bandwidth every clock tick. A request that gets completely served in the quanta leaves the system; otherwise it is cycled back into the queue for another quanta in the next clock tick. Fig. 2 depicts this architecture. Note that the quanta received by a request will vary every clock tick as new requests keep entering the system and completed requests leave.

3.3 Formal Statement

A data server DS_i is a collection of N_i data items $D_i = \{d_1^i, d_2^i, \ldots, d_{N_i}^i\}$ such that $D = \underset{i}{\cup} D_i$ and $D_i \cap D_j = \phi$ for $i \neq j$. The null intersection enforces the condition that the same data item is not hosted by two different data servers, i.e., no replication. A broadcast server maintains a dynamic queue with requests arriving as a tuple $\langle arr_Q, d_Q, dln_Q \rangle$, where arr_Q is the arrival time of a request Q, $d_Q \in D$ is the data item requested and $dln_Q > arr_Q$ is the absolute deadline for the request. Further, let b_{bs} and b_{ds} be the bandwidth available at the broadcast and data server respectively. A server with bandwidth b can transmit b data units per clock tick.

At each scheduling instance, the scheduler first removes all requests from the queue that will be served by the current broadcast and generates a schedule for all remaining requests Q'. A schedule is thus a total ordering of the data items in $\underset{Q'}{\cup} d_{Q'}$. Let $ft(d)$ be the time required to fetch the data item d from the corresponding data server. If t_{ready} is the ready time of the broadcast channel (the time instance when any ongoing broadcast ends), $d_{i_1} \rightarrow d_{i_2} \rightarrow \cdots \rightarrow d_{i_P}$ the broadcast schedule and s_d be the size of data item d, then the broadcast of an item d_{i_k} ends at time $t_{d_{i_k}} = t_{ready} + \sum_{j=1}^{k} [ft(d_{i_j}) + \frac{s_{d_{i_j}}}{b_{bs}}]$. All requests Q' served by this broadcast then has a completion time $ct_{Q'} = t_{d_{i_k}}$ and response time $rt_{Q'} = (ct_{Q'} - arr_{Q'})$. The objective of the scheduler is to generate a schedule such that the response time of the requests do not exceed the limit set by the deadline, i.e., $rt_{Q'} \leq (dln_{Q'} - arr_{Q'})$, or $ct_{Q'} \leq dln_{Q'}$.

3.4 Performance Metrics

We shall use three metrics – *deadline miss rate, average stretch* and *utility* – to evaluate the performance of different schedulers. Measurements are taken when all requests in the experimental data set have been served.

3.4.1 Deadline Miss Rate (DMR)

Let $\mathcal{Q}$ be the set of all requests served by the broadcast server at an instance of time. The deadline miss rate (DMR) is the fraction of requests in $\mathcal{Q}$ which missed their deadlines, given as,

$$DMR = \frac{|\{Q \in \mathcal{Q} | rt_Q > (dln_Q - arr_Q)\}|}{|\mathcal{Q}|}$$

3.4.2 Average Stretch (ASTR)

The stretch of a request is the ratio of its response time to its service time. Recall that the service time of a request is the time it would take to serve the request had it been the only one in the system. In a non-local data model, the service time should also include the retrieval time of the data item requested. Hence, if the request Q involves a data item of size s, to be retrieved from a data server with bandwidth b_{ds} and broadcasted over a channel with bandwidth b_{bs}, then the service time of Q is given by $st_Q = s(\frac{1}{b_{ds}} + \frac{1}{b_{bs}})$. The stretch of the request is then given as $STR_Q = \frac{rt_Q}{st_Q}$. A low stretch value indicates closer proximity of the response time to the minimum time needed to serve the request.

Intuitively, it seems that the response time of a request can never be less than the service time. Hence, the stretch of a request should always be greater than or equal to 1.0. However, when data fetching times are significant, the stretch can be less than one. This can happen when a new request for a data item next scheduled for broadcast, and being currently retrieved from a data server, enters the broadcast server. In such a situation, the request realizes a retrieval time less than $\frac{s}{b_{ds}}$ and gets served as soon as the retrieval ends. Therefore, the response time becomes less than the service time. The average stretch (ASTR) is given as,

$$ASTR = \frac{\sum_{Q \in \mathcal{Q}} STR_Q}{|\mathcal{Q}|} = \frac{1}{|\mathcal{Q}|} \sum_{Q \in \mathcal{Q}} \frac{rt_Q}{st_Q}$$

Ideally, a low ASTR signifies that most requests are served with minimal deviation from their service times. Hence, most requests will meet their deadlines as well, resulting in a lower DMR.

3.4.3 Utility (UT)

The performance level depicted by DMR is useful when the deadlines imposed are hard. However, when soft deadlines are in place, a request being served after its deadline still holds some utility. Thus, the utility (UT) metric uses a function that maps the response time of a request to a real number. We use the following function in this context.

$$U_Q = \begin{cases} 1 & , rt_Q \leq (dln_Q - arr_Q) \\ e^{-\alpha_Q(rt_Q - dln_Q + arr_Q)} & , otherwise \end{cases}$$

where $\alpha_Q = \ln 0.5/(dln_Q - arr_Q)$. Thus, the utility of a request is 1.0 if served by its deadline, otherwise decreases exponentially depending on the relative deadline of the request. The fractional utility attained in the system is then given as,

$$UT = \frac{\sum_{Q \in \mathcal{Q}} U_Q}{|\mathcal{Q}|}$$

An UT value of 1.0 indicates that all requests are served by their deadline; otherwise the closer it is to 1.0, the lesser is the time by which requests overshot their respective deadlines. Note that a high UT does not necessarily indicate a low DMR.

4 Schedule Generation

A major challenge in the generation of optimal schedules for data broadcasting is the lack of a formal theory underpinning the statistical characteristics of a "good" broadcast schedule. While queuing theory provides preliminary grounds for such analysis, no attempt is known to have been made to understand a broadcast system that reflects a one-to-many relationship between servings and arrivals in the queue. Hence, a significant amount of focus is concentrated in designing heuristic methods that employ intuitive perceptions of good broadcast mechanisms. Much of this is also due to the additional constraint of scheduling time imposed by the real time requirement in on-demand systems. A typical workload condition may prohibit the use of time consuming methods in order to avoid long accumulation of requests and increases in their response times. Long scheduling times may also keep valuable broadcast bandwidth idle, resulting in inefficient utilization of available resources. To this end, heuristic driven schedulers are often preferred as fast decision makers.

Most of the existing heuristics in broadcast scheduling have been evaluated assuming systems with local data availability. For this study, we adopted four well-performing heuristics – *RxW*, *MAX*, *SIN*-α and *PRDS* – proposed for such systems and observe their performance in non-local data availability systems. While no intrinsic property in these heuristics prohibit them from use in our problem model, modifications are made if required.

4.1 RxW

The RxW heuristic (Aksoy and Franklin, 1999) combines the benefits of the MRF (Most Requested First) and FCFS (First Come First Serve) heuristics in order to provide sufficient weight to both heavily requested and long awaited data items. Owing to its simplicity, the RxW heuristic has a low overhead in terms of scheduling time. RxW schedule 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.

4.2 MAX

The MAX heuristic (Acharya and Muthukrishnan, 1998) first assigns a hypothetical deadline to each request based on the maximum stretch value observed in the already served requests. For a request Q arriving at time arr_Q and with a service time st_Q, the hypothetical deadline is calculated as $(arr_Q + st_Q \times S_{max})$, where S_{max} is the maximum stretch observed till now in the served requests. Once the hypothetical deadline for all outstanding requests have been assigned, the MAX heuristic uses EDF (Earliest Deadline First) – the closer the hypothetical deadline to the current time, the higher the preference – to generate the schedule. Following the suggestions in the original work, the hypothetical deadline for a request is not changed once assigned even if the maximum stretch S_{max} is updated over time.

The MAX heuristic effectuates a scheduling mechanism that is targeted towards minimizing the maximum stretch value in the system. By doing so, MAX tries to maintain an optimal response time for requests, taking into account that different requests typically involve data items of different sizes. Note that the hypothetical deadline is not related to the actual deadline imposed on a request. Nevertheless, if MAX manages to generate schedules that prohibit the maximum stretch from increasing drastically, then requests will be served with minimal deviations from the minimum time required to serve them. Assuming that actual deadlines are imposed reasonably, the requests are then likely to complete within their deadlines.

4.3 SIN-α

The SIN-α (Slack time Inverse Number of pending requests) heuristic (Xu et al., 2006) integrates the "urgency" and "productivity" factors into a single metric called $sin.\alpha$. The intuition behind the heuristic is explained by the authors using 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. The parameter α (≥ 0) can amplify or reduce the significance of the productivity factor while making scheduling decisions. With $sin.\alpha$ values assigned to the data items, the schedule is created in increasing order of the $sin.\alpha$ values. Note that the SIN-α heuristic does not take into account the different sizes of the data items involved in the requests. Hence, the heuristic does not differentiate between equi-probable data items with very different sizes. 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.

4.4 NPRDS

Apart from the two arguments provided for SIN-α, the PRDS (Preemptive Request count, Deadline, Size) heuristic (Lee et al., 2006) 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. Preemptive schedules are usually expected to perform better than non-preemptive ones. However, the mechanism has been discarded in this study to facilitate a fair comparison. Besides, preemptive scheduling in a non-local data model would either require the broadcast server to repeatedly request the same data item for the same set of requests (thus adding to the response time of the requests), or has to buffer all preempted data items. NPRDS 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. A higher value of this priority estimate indicates that the data item is waited for by more number of requests, can help attain a tighter deadline and will not take much time to broadcast. Hence, the NPRDS schedule is generated in decreasing order of the priority values. The NPRDS heuristic aims at providing a fair treatment to different data sizes, access frequency of data items and the deadline of requests.

5 Scheduling with Stochastic Information

The heuristics outlined in the previous section do not utilize any information available on the response times of the data server. This information is assumed to be available in the form of a probability distribution. Next, we propose two novel heuristics that utilize such stochastic information to compute a completion time distribution for data items. Thereafter, scheduling decisions are made based on the probability of serving a request within its deadline.

5.1 MDMP

The MDMP (Minimum Deadline Meet Probability) heuristic can be considered a stochastic version of EDF. Under MDMP, the highest preference is given to the data item corresponding to the request that has the minimum probability of meeting its deadline. In order to compute this probability, we first have to determine the completion time distribution of broadcasting a data item. In a situation where the response time of fetching a data item is deterministic, the completion time of a broadcast would simply be the addition of the ready time of the broadcast channel, the response time of fetching the data item and the time required to broadcast the item over the channel. A similar method is used for the case when the response time of fetching an item is stochastic. We begin with the response time pmf of the data server where the data item resides and combine it with the ready time distribution of the broadcast channel. Let $D_S(d)$ denote the data server where the data item d resides and X_k denote the random variable representing the response time of fetching a data item from data server k. If X_{bdst} denote the random variable representing the ready time distribution of the broadcast channel, then the completion time distribution of broadcasting data item d of size s_d is given by $X_{bdst} + X_{D_S(d)} + \frac{s_d}{b_{bs}}$. Hence, determining the completion time distribution requires us to find a way of adding random variables. This can be achieved by the convolution, denoted by $\odot$, of the pmfs of the corresponding random variables. Addition of a scalar to a random variable, denoted by $\oplus$, is equivalent to shifting the time axis of the random variable's pmf by the scalar amount. With these two operations, we can define the MDMP heuristic as follows.

Note that the convolution of X_{bdst} with the data server response time pmfs can be precomputed before step 3 in order to reduce the time complexity of the heuristic. Also, proper resolution of ties may serve to be crucial when there exists multiple requests in the queue that have already missed their respective deadlines, i.e. $Pr(X_Q \leq dln_Q) = 0$. Hence, instead of arbitrarily choosing a data item from equal deadline meet probability requests, we choose the one that can serve a higher number of requests and reduce request accumulation. Any ties happening at this level are broken arbitrarily. The MDMP heuristic gives the highest preference to the urgency factor in scheduling, followed by productivity.

Heuristic 1 Minimum Deadline Meet Probability

```
1. Let $t_{cb}$ be the time when the currently ongoing broadcast started and $s_{cur}$ be size of the item being broadcast.
2. Initialize $X_{bdst}$ such that $Pr(X_{bdst} = t_{cb} + \frac{s_{cur}}{b_{bs}}) = 1$. If no broadcast is currently taking place, then the initial pmf for $X_{bdst}$ contains a single impulse at the current time, i.e. $Pr(X_{bdst} = current\,time) = 1$.
3. For each pending request $Q$, compute $X_Q = (X_{bdst} \odot X_{D_S(d_Q)}) \oplus \frac{s_{d_Q}}{b_{bs}}$ and choose the request $Q^*$ such that $Pr(X_{Q^*} \leq dln_{Q^*}) = \min_Q Pr(X_Q \leq dln_Q)$; ties are broken by selecting the request which involves a data item that can serve a higher number of requests.
4. Schedule $d_{Q^*}$ as the next broadcast item. Mark all requests that would be served by $d_{Q^*}$ as "served".
5. Set $X_{bdst} = X_{Q^*}$.
6. Repeat from step 3 until all pending requests are marked "served".
```

5.2 MBSP

The MBSP (Maximum Bounded Slack Probability) heuristic can be visualized as the dual of MDMP with slight modifications. *Slack* in this case is defined as the duration from the deadline of a request to the time when it gets served. This slack value is positive if the request is served before the deadline, otherwise negative. MBSP schedules data items based on a lower bound for the slack value of requests. Since negative slack requests have already missed their deadlines, it might seem reasonable to give preference to ones which can still meet their deadlines. However, such a strategy can result in certain requests being pushed too far away from their imposed deadlines. Hence, MBSP employs a *slack deviation* parameter $\mathcal{S}_{dev}$ to extend the deadline of a request by a variable amount and measures the slack from this extended deadline. This is equivalent to imposing a lower bound on the slack value of requests. If a request misses this extended deadline too, then one can say that the request has been ignored for a long duration of time (after it missed the deadline) and should now be served. Thus, MBSP gives preference to requests with the maximum probability of missing the deadline extended by the slack deviation parameter, or in other words, to requests with the maximum probability of going below the lower bounded slack value. The heuristic can be computed as follows.

The $\ominus$ operator in step 2 signifies the subtraction of a random variable X from a scalar quantity q_{scalar}. This operation results in a random variable $X' = q_{scalar} \ominus X$, such that $Pr(X' = q_{scalar} - x) = Pr(X = x)$. Note that if $\mathcal{S}_{dev} = 0$, then MBSP simply chooses the request with the maximum probability of missing its deadline. We have used the relative deadline of a request as the factor to scale in deciding the extended deadline. This is primarily based on an understanding of the utility derived from serving a request that has already missed its deadline. Since we have assumed here that the drop in utility is related to the time overshot from the deadline, we adhere to this value in

Heuristic 2 Maximum Bounded Slack Probability

```
1. Initialize $X_{bdst}$ as in MDMP.
2. For each pending request $Q$, compute $X_Q = dln_Q \ominus [(X_{bdst} \odot X_{D_S(d_Q)}) \oplus \frac{s_{d_Q}}{b_{bs}}]$ and
   choose the request $Q^*$ such that $Pr(X_{Q^*} \leq \mathcal{S}_{dev} \cdot [dln_{Q^*} - arr_{Q^*}]) = \max_{Q} Pr(X_Q \leq$
   $\mathcal{S}_{dev} \cdot [dln_Q - arr_Q])$; ties are broken by selecting the request which involves a data item
   that can serve a higher number of requests.
3. Schedule $d_{Q^*}$ as the next broadcast item. Mark all requests that would be served by
   $d_{Q^*}$ as "served".
4. Set $X_{bdst} = X_{Q^*}$.
5. Repeat from step 2 until all pending requests are marked "served".
```

calculating the extended deadline as well. Other representations of utility may signify a different formulation to be more appropriate.

The novelty in MDMP and MBSP lies in the treatment of the stochastic information available about retrieval times of data items. A typical scheduler can compute the expected value from the response time distribution and use it as the time required to fetch any data item from the data server. Therefore, the scheduler assumes that a data item of any size can be fetched in the same amount of time, and ignores the varying workload in the data server. MDMP and MBSP work directly with the probability distribution and hence utilize convolution, instead of direct sums of broadcast and expected retrieval times, to find completion time distributions. The potential advantage of obtaining this completion time distribution lies in the fact that a scheduler can now compute the exact probability with which a particular request will be completely served within a given time period. To our knowledge, using the probability of successfully completing a broadcast to build a schedule has not been attempted earlier, and is an approach that diverges from typical deterministic methods.

6 Experimental Setup

The experimental setup used in this study consists of a single broadcast server and four data servers. Requests at the broadcast server arrive at a rate of $\lambda_{bs} = 5$ requests per second following a Poisson distribution. Each request consists of an arrival time, a data item number and an absolute deadline. A total of $100,000$ such requests are generated for inclusion in a synthetic data set. A data item is chosen for a request based on its size and popularity, as discussed in Sections 6.1 and 6.2.

We have written our custom tool to simulate the broadcast and data server interactions. The tool is a sequential control flow program that switches between a *broadcast server module* and a *data server module*, and measures the various time related statistics required in the performance metrics. Note that although our implementation has a sequential control flow, the execution of the data server module is designed in a way that the data servers appear to be running in parallel to the broadcast server. Details of this implementation is given in Section 6.7. Owing to the nature of the implementation, a single machine is sufficient as a host for the tool to measure the relevant time statistics (completion time of requests). Different data servers are only instantiations of the data server module.

6.1 Data Item Popularity

Data items are requested following the commonly used *Zipf*-like distribution (Breslau et al., 1999) with a characterizing exponent of $\beta = 0.8$. A larger exponent means that more requests are concentrated on a few hot items. Breslau et. al found that the distribution of web requests from a fixed group of users can be characterized by a *Zipf*-like distribution with the exponent ranging from 0.64 to 0.83. They also found that the exponent centers around 0.8 for traces from homogeneous environments. In order to use such a distribution, data items are first assigned *ranks* which are then used in deciding the probability of requesting a data item of a particular rank. The probability of choosing a data item of rank k is given as,

$$Pr(Rank = k) = \frac{1/k^{\beta}}{\sum_{n=1}^{N}(1/n^{\beta})}$$

where N is the total number of data items. A total of 400 data items is used here. Ranks are assigned to these data items and the probability distribution is used to determine which item is requested as part of a request. The rank 1 item is the most frequently requested, while the rank 400 item is least requested. Note that we still need to assign a size to each data item.

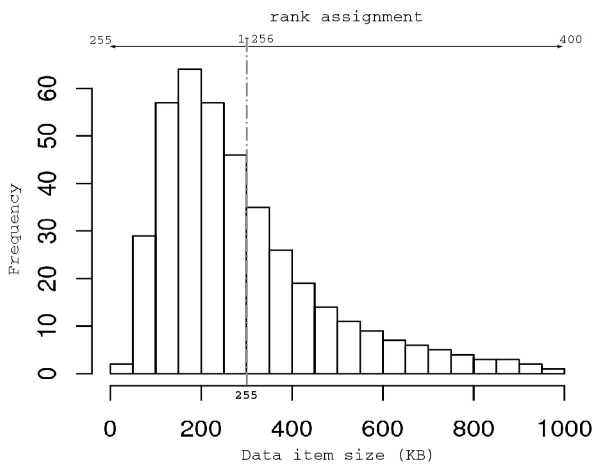

Fig. 3 Data item size distribution and rank assignment.

6.2 Relating Data Item Size to Popularity

Data item sizes vary from $5KB$ to $1MB$ in increments of $50KB$. The data sizes are set based on the assumption that most data items appearing in relevant application domains will typically contain raw data or encoded using some markup language. A data item of $500KB$ in itself can contain significant amounts of information in this case. We do assume that some large files may exist and hence set the upper limit at $1MB$. These values also reflect the numbers used in related literature. Assignment of sizes to data items of different ranks is done using a hybrid distribution (Barford and Crovella, 1998). The body of this distribution follows *lognormal* characteristics while the tail follows a heavy tailed *Pareto* distribution, given as

$$Pr(Size = s) = \begin{cases} \frac{1}{s\sigma\sqrt{2\pi}} e^{-(\ln s - \mu)^2/2\sigma^2} & , s \leq s_{break} \\ \alpha k^{\alpha} s^{-(\alpha+1)} & , s > s_{break} \end{cases}$$

where the parameters are set as follows: $\mu = 5.5$, $\sigma = 0.6$, $\alpha = 1.4$, $k = 175$ and $s_{break} = 500$. The parameters are set so that the lognormal distribution spans over the $0-500KB$ range and smoothly transitions into the Pareto distribution from $500KB$ onwards. Fig. 3 shows the data item size distribution and the rank assignment scheme on the items. As also observed in web traces used by Barford and Crovella, about 82% of the data items lie in the body of the distribution with the used parameter settings. Note that an actual workload may demonstrate slightly different values for these parameters depending on the number of items, minimum and maximum data size, access frequencies of items, etc. However, the overall distribution is expected to remain consistent with the one used here. The performance results are therefore not restricted to the used parameter settings, but are rather dependent on the effectiveness of the underlying distributions in capturing real workload characteristics.

Each unique data item with size from $5KB$ to $255KB$ are assigned ranks from 1 to 255 in descending order of the sizes. This ranking scheme continues from 256 to 400 with data items of size higher than $255KB$ in ascending order. The rank assignment makes sizes in the range $[5KB, 255KB]$ more frequently requested, while bigger sizes get a below average request frequency. The average file size under this distribution is $s_{avg} = 205KB$. This setup follows the general observation made by Breslau et. al that the average size of less frequently accessed data items is comparatively larger than that of frequently accessed items (Breslau et al., 1999).

6.3 Assigning Deadlines to Requests

Expected response times are assigned to requests from a normal distribution with mean $60s$ and standard deviation $20s$. This response time is added to the arrival time of a request to get the absolute deadline for the request.

6.4 Varying Workloads

Heuristic performance can greatly vary depending on the workload at the broadcast server. One way to simulate different workload conditions is by changing the request arrival rate. Higher request rates result in a larger number of pending requests to schedule at a given time instance. The scheduler's performance is then gauged by how fast the request queue can be cleared by the schedule built by it. This assessment assumes a fixed (and reasonable) bandwidth available for broadcast. An alternative to this, and the one adopted here, is to modulate the bandwidth *utilization factor* of the broadcast server (UF_{bs}) and keep the request rate fixed. By changing the utilization factor we can effectuate different bandwidths available at the broadcast server. This in turn changes the rate at which a particular schedule can be executed, thereby modulating the number of pending requests and simulating different workload conditions. The bandwidth is thus assigned as $b_{bs} = \lambda_{bs} s_{avg} / UF_{bs}\ KB/s$. Given a fixed bandwidth b_{bs}, the request rate is directly proportional to the utilization factor.

6.5 Generating Requests at Data Servers

The data servers are simulated as $M/G/1$-PS systems running in parallel with the broadcast server. The bandwidth available at a data server is fixed at $b_{ds} = 5760 KB/s$ (a T3 connection). The 400 data items are randomly assigned to the four data servers. Every request coming to a data server has an associated service time (time required to serve the request had it been the only one in the system). For requests originating at the broadcast server, this service time is equal to the size of the data item to be retrieved divided by the total bandwidth at the data server. The data servers are also accessed by clients other than the broadcast server in our simulation setup. Service times for requests from such clients are generated as follows. First, a sample of 10 million requests is generated for the broadcast server. For each data server, the sizes of data items in the sample that would be served by this server are then used to generate a size distribution for the server. The service time distribution for the server follows this size distribution, with service times given as the size values divided by the bandwidth. Every time a new request is to be generated at the server, the service time distribution is sampled and the value is used as the service time of the request. The size distribution also lets us calculate an expected data item size that is requested at the server. Requests arrive at a data server at the rate of $\lambda_{ds} = 16$ requests per second according to a Poisson distribution. This, coupled with the expected data item size information, generates a 70% bandwidth utilization in the data servers.

6.6 Estimating Data Server Response Time Distributions

Recall that a data server distributes its available bandwidth equally to all pending requests in a round-robin fashion. Hence, depending on the number of requests pending at a server, the time required to retrieve a data item from the server, i.e. the response time of the data server, would typically be higher than the service time of the request. Application of the proposed methods will require an approximation of the probability distribution of data retrieval times. We do not assume any known probability distribution. Rather, we suggest that the approximation be obtained by sampling the data server, and representative points be used to obtain the approximated pmf. In order to generate the response time pmfs of the data servers, each server is run independently and the response times of 10 million requests are noted. This gives us an estimate of the response time distribution at the server, from which 100 points are chosen uniformly to construct the pmf. Constructing the pmfs in this manner also allow the methods to switch to a different distribution depending on other external factors (e.g. time of day).

6.7 Software Implementation

Note that the data servers run independently of each other and the broadcast server. Hence, the request queue in the data servers will continuously change (depending on its own requests rate and response times) irrespective of whether requests are coming from a particular broadcast server or not. The broadcast server requests an item from a data server only when the current broadcast ends and there is an item waiting for broadcast next in the schedule. We pay necessary attention to maintain the dynamics of the data server during the time when the broadcast server is in the process of broadcasting an item. Our implementation achieves this using sequential programming of two components - (i) the *broadcast server module* (BSM) and (ii) the *data server module* (DSM). A system global *Clock* keeps track of the current time (initialized to zero). The BSM reads the data set file and inserts all requests having an arrival time equal to *Clock* into a request queue. The scheduling algorithm is then invoked to generate a schedule to serve the requests in the queue. *Clock* is incremented by one and portions of the schedule that would get executed within this time period (between the old and new value of *Clock*) is updated. The update is performed as follows. The data server hosting the data item next scheduled for broadcast is determined. The DSM for the corresponding data server is called to compute the retrieval time for the data item. It executes by infusing its request queue according to a Poisson distribution, with service times drawn as described in Section 6.5. Each DSM maintains its own local clock that runs at 1000 ticks per *Clock* tick. The data item request of BSM is inserted into the queue only when the DSM's local clock is at the time when the previous broadcast (at the broadcast server) ended. This guarantees that the data server is running even when no request was made by the broadcast server. The DSM returns back to the BSM as soon as the requested data item is completely served. In addition, it saves its current state (queue and local clock) if the local clock time did not surpass *Clock*. Similarly, the BSM considers the broadcast of the data item complete only if the retrieval of the item did not overshoot into the next clock tick. A completed broadcast implies removal of all requests in the queue waiting for the data item. The BSM reads the next set of requests in the data file and repeats the process.

DSMs corresponding to the four data servers are initialized by running them for at least 50000 clock ticks before starting the BSM.

6.8 Convolution Time

Convolution is implemented using an $O(n \log n)$ algorithm with *Fast Fourier Transforms* (Press et al., 1992). Iterative convolution, as in MDMP and MBSP, can increase the convolution time as the number of samples in the generated pmf (X_{bdst}) increases. In general, if pmf f has N_f samples and pmf g has N_g samples, then their convolution contains $N_f + N_g - 1$ samples. When applied iteratively, the pmf size can increase considerably, thereby increasing the convolution time as well. Hence, in order to keep the scheduling time within acceptable limits, we direct the algorithm to generate resulting pmfs with a maximum of 100 samples.

6.9 Other Specifics

All experiments are run on a 1.86GHz Intel Core$^{\text{TM}}$ 2 Duo system with 2GB memory and running Fedora Core 8. The slack deviation parameter $\mathcal{S}_{dev}$ is set at 0.25, unless otherwise stated. The scheduler is invoked every second until the last request in the data set enters the queue; thereafter, it is invoked every five seconds until the request queue becomes empty. The scheduler is invoked every second since the schedule built by it is based on probabilistic estimates of retrieval times of data items (especially for MDMP and MBSP). The actual retrieval time is known only after a data item is fetched completely from the data server, in which case, the scheduler may require to rearrange the remaining part of the schedule for a better QoS. This is also the reason why the scheduler is repeatedly invoked even if no new request enters the queue. Our implementation operates in a batch mode, meaning, the current schedule is revisited (and reorganized if required) after a few data items (more precisely, as much as that can be fetched and broadcasted in one tick) have been fetched and broadcasted. Ideally, the scheduler is a continuously running process in the system. In a real world setting, the execution of the scheduler is primarily decided by the priority of other processes running in the system.

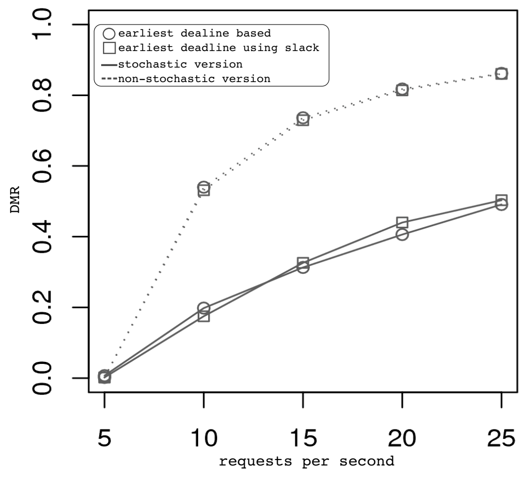

Fig. 4 Comparative deadline miss rates of identical algorithms in their stochastic (using probability distributions) and non-stochastic (using expected values) versions.

7 Empirical Results

In this section, we present simulation results for the six heuristics. Performance measurement is taken once all 100,000 requests in the data set are served. The total time spent in scheduling instances between a request's arrival and completion is added to the response time of the request while taking the measurements. As we know, broadcast performance is heavily influenced by various factors such as the bandwidth, parameter settings in an algorithm, data size, their frequency of access and, in overall, by the broadcasting policy resulting from them. Results pertaining to the impact of such factors are also discussed here. Our experiments also reveal counter-intuitive observations in the ASTR metric. A possible explanation of this behavior is also discussed here.

7.1 Effectiveness of Using Distributions

In order to demonstrate the effectiveness of using data server response time distributions in scheduling, we compare the performance of MDMP and MBSP to their non-stochastic versions. The non-stochastic versions use the expected value of the response time distributions as an estimate of the time required to fetch a data item from a server. The non-stochastic version of MDMP serves requests in ascending order of their closeness to the corresponding deadlines. The expected response time is used to compute when a particular data item broadcast will end, depending on which the earliest deadline request from the pending queue is determined. A similar methodology is used for the non-stochastic version of MBSP, the difference being that the extended deadline (as determined by the slack deviation parameter) is used instead of the actual one.

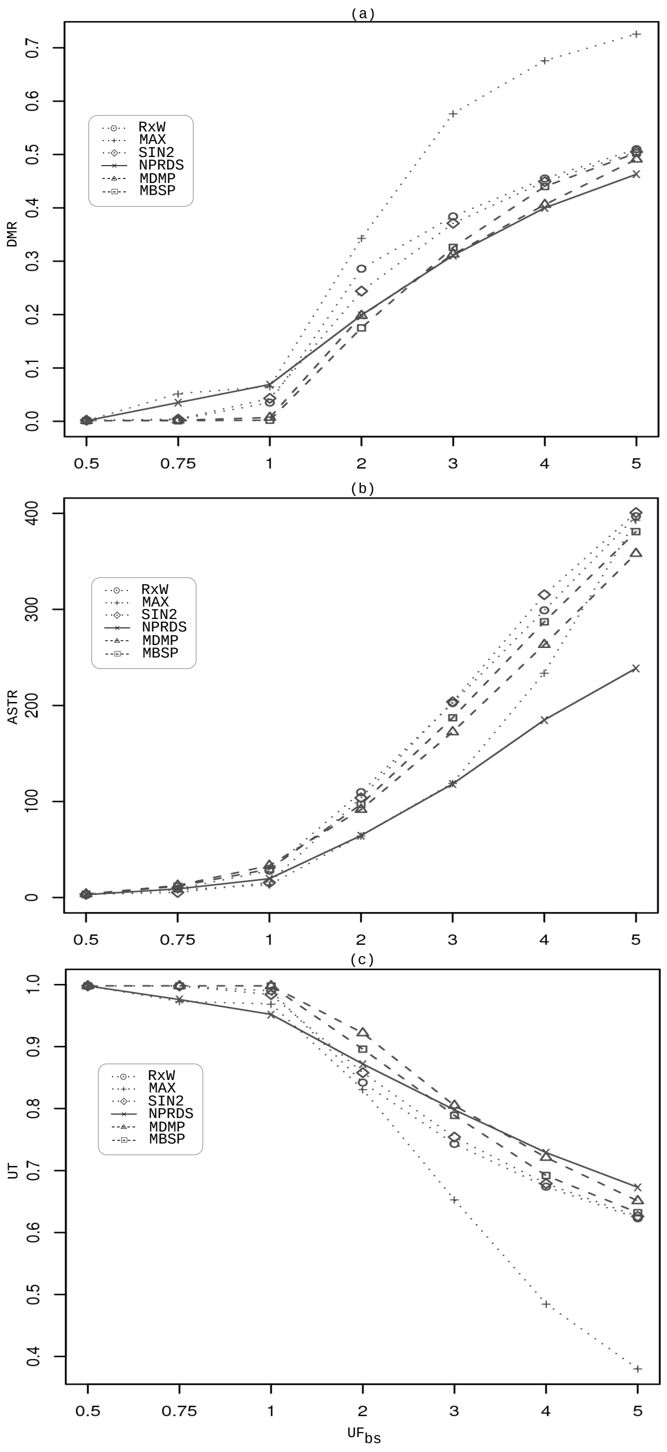

Fig. 5 Performance metric values for different heuristics under different workloads.

Fig. 4 depicts the DMR of the heuristics for varying request arrival rates with a fixed broadcast bandwidth of $1MB/s$. While differences are marginal within the stochastic and non-stochastic versions, significant improvements can be observed when creating schedules based on the probability distributions directly. Deadline misses in the non-stochastic versions are almost double that of the stochastic versions as the workload on the broadcast server increases. This is not a surprising observation since the non-stochastic versions work under the assumption that all data items can be fetched in a fixed amount of time irrespective of the varying workload on the data server. The characteristics of the data item sizes hosted at a data server and the potential impact of the data server workload on their retrieval are more thoroughly maintained in the distributions. The non-stochastic versions do not have a direct method to utilize this information and essentially treats the data servers as a single client secondary storage with constant access time. Note that the expected value is the average response time over a fairly large number of requests made to a data server. However, requests from a broadcast server to a given data server may be sporadic depending on the popularity of the data items hosted by the server.

7.2 Comparative Performance

Fig. 5 shows the performance metric values for the six heuristics under different workload conditions. Variable workload conditions are simulated by changing the bandwidth of the downlink channel, as given by the different bandwidth utilization factors $UF_{bs} = 0.5, 0.75, 1.0, 2.0, 3.0, 4.0, 5.0$. An utilization factor greater than 1.0 means data requests arrive at a rate higher than that can be handled by the broadcast server bandwidth. With the fixed bandwidth model, these factors translate to workloads generated by more than 5 requests per second with $b_{bs} = 1MB/s$. MDMP and MBSP maintain a DMR of less than 1% as the bandwidth utilization approaches 100%. On the other hand, the DMR of NPRDS increases to 8%. In general, deadline miss rates are less than 10% for utilization factors of less than 1.0. Similarly, all heuristics can generate schedules with more than 95% utility at such utilization factors. All heuristics show an exponential increase in the number of deadlines missed when the utilization factor goes above 1.0. The drop in utility and increase in the average stretch follow similar trends. Both stochastic heuristics, MDMP and MBSP, perform better in DMR and UT over a wider range of bandwidth utilization. MDMP and NPRDS have around 20% deadline misses at $UF_{bs} = 2.0$, compared to 34% of MAX. The utility metric is between 82% (MAX) and 93% (MDMP) at this point. However, as mentioned earlier, the average stretch metric does not reflect this performance. MAX and NPRDS are the better performing ones according to ASTR. We also notice an improvement in the comparative performance of NPRDS as bandwidth utilization goes beyond 300%. The NPRDS heuristic displays better sustenance in all three metrics at such utilizations.

7.3 Impact of Broadcast Utilization Factor

Heuristic performance is heavily affected by the workload conditions and bandwidth available at the broadcast server. In conditions of heavy workload, or low bandwidth, the accumulation of requests at the broadcast server dominates the rate at which requests get served. If frequently requested items are not regularly broadcast, the request queue gets flooded with such requests. On the other hand, if they are broadcast too often, then the accumulation happens for average and less frequently requested data items. However, we believe the latter factor plays a more prominent role. This is because, although requests for frequent items can get accumulated in the queue, a broadcast policy can let this accumulation happen and reduce the queue size considerably by a single broadcast of the item. This strategy will not work for average and less frequently requested items since the accumulation of such requests will typically take a longer time resulting in a number of them missing their deadline when the broadcast is finally made. In low workloads, most requests are typically for frequently requested items and hence a relatively smaller number of broadcasts can serve most requests in the queue.

Irrespective of the broadcast policy generated by a heuristic, deadline miss rates can still increase with decreasing bandwidth. This is in general attributable to the higher time required to transmit the data items of even an average size. Thus, requests that are supposed to be served by data items towards the end of the schedule suffer a high response time. The situation worsens if an intermediate broadcast is for a larger item.

MDMP and MBSP maintain lower deadline misses for utilization factors less than 3.0. NPRDS and MAX have the worst DMR for $UF_{bs} \leq 1.0$. Recall that NPRDS is the only heuristic that considers the size of the data item in its scheduling decisions. This makes us believe that the size of a data item is not an important factor to consider when scheduling for a high bandwidth system. Clearly, the consideration of size becomes important when bandwidth is low as depicted by the better performance of NPRDS. Unlike the size of data items, stochastic information utilized by MDMP and MBSP seem to become less and less relevant for higher UF_{bs}. The probability estimates do help in sustaining the performance of the two heuristics beyond the $UF_{bs} = 1.0$ mark (note that NPRDS overtakes most other heuristics at this point). To our understanding, for very low broadcast bandwidth, the retrieval time of data items (carried out through a high-speed connection) becomes negligible in comparison to their broadcast times, which in effect diminishes any improvements obtainable by accounting for the retrieval time in scheduling decisions.

An interesting observation to note is at $UF_{bs} = 2.0$. Although, the DMRs of NPRDS, MDMP and MBSP are quite similar at this point, there is still a significant difference in their UT values. NPRDS and MDMP displays a DMR of about 20% and MBSP has a DMR of 17.5%. However, MDMP maintains a higher utility than NPRDS (93% compared to 87%). The requests which missed their deadlines in MDMP did so by smaller durations than in NPRDS. Hence, equivalent deadline misses need not always correlate to similar utilities.

Broadcast schedules can be generated that, although misses equal number of deadlines, can maintain closer proximity of the response time of requests to their service times. MBSP, on the other hand, displays a trade-off characteristic where lower DMR has been achieved by a slight reduction in the utility (89.6%).

Based on these observations, MDMP and MBSP are probable choices for low and marginally higher workloads, while NPRDS is the likely candidate for very heavy workloads. One should note that deadline misses will increase exponentially when bandwidth utilization goes higher than 100%. Marginal increases in the utilization factor are acceptable owing to bursts in traffic. However, the range of utilization factors where heuristics such as NPRDS performs better in this problem are very unlikely settings for a realistic broadcast system (a system having almost 50% deadline misses). If a situation does occur where the utilization surpasses the expected range, then the network design is revisited or additional resources are installed to bring down the utilization to the sought range. In this regard, MDMP and MBSP both sustain their better performance up to 300% utilization. The observation helps conclude that MDMP and MBSP are also suitable for handling occasional bursts in traffic.

Owing to the differences in performances of the heuristics in light and heavy workload scenarios, it may appear that switching the scheduling algorithm accordingly is an alternative to obtaining better quality of service (QoS). However, such an approach requires the considerations of other issues. A burst in traffic is usually short-term. Switching of algorithms for such short periods can result in an abrupt change in the QoS properties of the system. It is also not likely that the algorithm switched to can return the sought improvement in overall QoS in a short duration of time. Note that the performance results reported here assume that a particular heuristic is the sole one executing in the system. It is not known whether the same performance would be achievable when the heuristics are dynamically changing. Other factors such as the switching cost and the latency induced owing to changes in the current schedule must also be accounted. It remains to be investigated if the dynamic management of scheduling algorithms can improve the performance of a broadcast system at all.

7.4 Impact of Slack Deviation on MBSP

The MBSP heuristic first schedules the request with the maximum probability of missing the extended deadline decided by the slack deviation parameter $\mathcal{S}_{dev}$. The extended deadline for a request Q is given as $dln_Q + \mathcal{S}_{dev}(dln_Q - arr_Q)$. Table 1 shows the variation in the performance metrics across five different slack deviations. Recall that $\mathcal{S}_{dev} = 0$ corresponds to the heuristic that prefers the request with maximum probability of missing the deadline and shows very similar performance to the dual heuristic MDMP (minimum probability of meeting the deadline). While ASTR and UT show consistent degradation as $\mathcal{S}_{dev}$ is increased from 0.0 to 1.0, DMR displays a change in gradient around $\mathcal{S}_{dev} = 0.25$.

$\mathcal{S}_{dev}$	DMR	ASTR	UT
0.00	0.198	93.0	0.922
0.25	**0.175**	97.0	0.896
0.50	0.236	111.8	0.866
0.75	0.263	124.9	0.84
1.00	0.266	132.5	0.833

Table 1 Performance metric values for MBSP with $UF_{bs} = 2$ and varying $\mathcal{S}_{dev}$.

The intuition behind MBSP is to provide "sufficient" slack to the scheduler to satisfy other requests at the expense of ones which are likely to or have already missed their deadlines. This is achieved by informing a longer deadline for requests to the scheduler, instead of the actual one. The slack deviation parameter helps modulate the amount of slack so that exploited requests do not remain in the system for ever. The key to setting the parameter is understanding what serves as "sufficient".

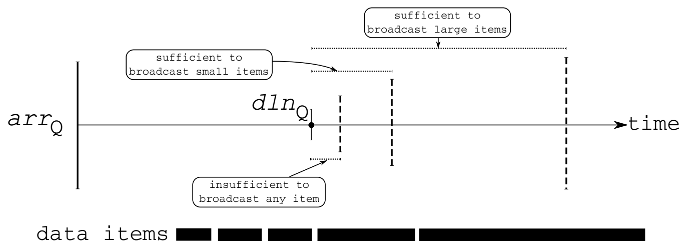
sufficient to broadcast large items
sufficient to broadcast small items
dln_Q
arr_Q
time
insufficient to broadcast any item
data items

Fig. 6 Impact of $\mathcal{S}_{dev}$ on scheduling decisions.

Refer to Fig. 6. When the slack deviation is too small, the slack is of little or no help since no data item (to serve other requests) can be broadcast between the actual and the extended deadline. The performance of MBSP would thus be equivalent to that of MDMP. The reason why a value such as 0.25 works best for the tested workloads can be attributed to the size of the most frequently requested items (an item of around $200KB$ in the experimental setup). Assume that a request Q has already missed its actual deadline $t = dln_Q$. Further, let its extended deadline be $t + x$, where x is determined by the slack deviation parameter as $x = \mathcal{S}_{dev}(dln_Q - arr_Q)$. Let the current time be $t+1$. Since MBSP sees the deadline for Q as $t+x$, it would try to schedule other requests first (ones which are closer to their extended deadlines) provided the probability of Q missing its extended deadline is not the maximum. If x is small enough (correspondingly a value such as 0.25) that only moderately sized data items can be fit in the schedule before Q's probability of missing the extended deadline becomes the maximum, then there will be a higher likelihood that a more frequently requested item is scheduled next. This can be viewed as a positive effect of using the slack since many requests will get served by this broadcast. However, for larger x, broadcast of a large data item can also be accommodated within the slack period. This is a negative effect of using slack since broadcasting such an item would serve only one or two requests, in addition to delaying the other pending requests further. The sufficiency of the slack is thus related to the size of data items that are more frequently requested. Intuitively, the slack deviation should be set so that the introduced slack is not misutilized in broadcasting data items that would serve very few requests. Of course, this explanation is valid only in the context of our experimental setup. In our setup, the infrequent data items were the ones larger than $500KB$. Hence, higher slack values demonstrated poorer performance. On the other hand, if frequently requested items are indeed the larger data items, then introducing a higher slack would be beneficial.

A favorable value for the slack deviation parameter may be approximated using a hill climbing search in real time. Such an approach can intermittently change the parameter value until desired service levels are obtained or no further improvements are observed. The initial value can be set based on the characteristics of frequently requested items as discussed before. This estimation process lends itself to the possibility of changes in the value as the request characteristics change over time. Note that the optimal value for slack deviation is sensitive to the size of the frequently requested items. Therefore, changes in the request characteristics will also influence the optimality of the slack deviation parameter. Although hill climbing cannot guarantee global optimality, it is an efficient alternative to randomly deciding a value for the parameter.

7.5 Effect of Scheduling by Size

In this section, we shall try to understand the effect of considering the size of data items in scheduling decisions for different workload conditions. NPRDS is the only heuristic we explored that explicitly uses data size in its formulation. Fig. 7 illustrates the frequency distribution of broadcasts by the rank of data items. At $UF_{bs} = 0.75$, RxW and SIN2 has a DMR value of less than 0.005, while that of NPRDS is 0.035.

The broadcast policy adopted by RxW and SIN2 results in a frequency distribution similar to the Zipf distribution of data item requests. NPRDS shows peaks in the range of data items that are requested on a more than average basis. As seen in the plot, even closely ranked data items can have very different broadcast frequencies.

Both RxW and SIN2 consider only the urgency and productivity factors, while NPRDS also considers the size in addition. The scheduling policy used by NPRDS is based on a priority value that is higher for smaller sized data items (assuming similar productivity and urgency). Recall the rank assignment scheme for data items. Most average requested data items under the scheme

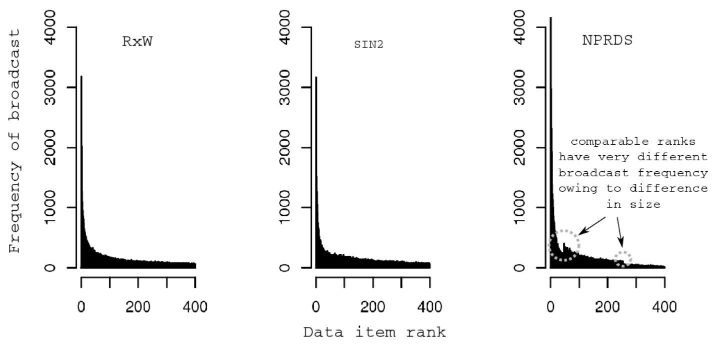

Fig. 7 Broadcast frequencies of data items by rank at $UF_{bs} = 0.75$.

have lower (but comparative) data size than the frequently requested items (in the range of $155KB$ to $205KB$). Contentions are not unlikely between data items of similar sizes. Consider a case where a request for a data item of size $255KB$ and a request for a data item of size $155KB$ are equally urgent. Let there be 30 requests for the $255KB$ item and 20 for the $155KB$ item. RxW and SIN2 in this case would give preference to the $255KB$ item, whereas NPRDS would choose the $155KB$ item ($\frac{20}{155} > \frac{30}{255}$). Note that the difference in broadcast time of the two items will be very small in a high bandwidth system. Hence, broadcasting the data item that can serve more requests (the $255KB$ item) would be more sensible so that less number of requests (the ones for the $155KB$ item) have to wait further. However, the policy of NPRDS can be useful in low bandwidth situations since broadcast of the smaller item achieves the objectives of inducing lower latencies for other requests and serving a significant number of requests. The better performance of NPRDS in heavy load conditions conforms to this justification. Thus, size considerations are more important when fast servings are required for a large number of requests in a relatively small amount of time.

7.6 Stretch and Performance

Recall that the MAX heuristic has the worst performance according to DMR and UT across all variations of the utilization factor. This performance indication contradicts the picture presented by the ASTR metric. MAX maintains lower average stretch across a wider range of utilization factors. We also observe that the maximum stretch in a MAX schedule is comparatively much lower than that from the other heuristics. Nevertheless, MAX misses the highest number of deadlines. Ideally, lower stretch values mean that response time of requests are closer to their service times, and hence they should be able to meet their deadlines. To understand this discrepancy, we refer to the distribution of the individual stretch values under MAX and MDMP generated schedules.

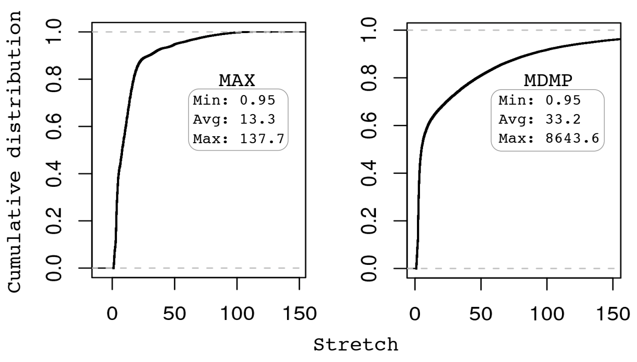

Fig. 8 Cumulative probability distribution of stretch values at $UF_{bs} = 1.0$.

Fig. 8 shows the cumulative probability distribution of stretch values for MAX and MDMP at $UF_{bs} = 1.0$. At this utilization, MAX has a DMR of 6.5% while that of MDMP is 0.7%. However, the average and maximum values of stretch is much lower in MAX. In fact, MAX demonstrates clear efficiency in keeping the maximum stretch of the system at a minimum at all levels of utilization. The distribution reveals that the stretch values of about 60% of the requests are very similar across the two heuristics. Sharp differences appear in the remaining 40%. These differences shifted the average stretch of the system to a higher value in MDMP. However, it still does not explain why MAX has a higher DMR. Since the stretch values are spread over a much wider range for the aforementioned 40% of the requests in MDMP, the likelihood of missing deadlines is more than MAX for such requests. Thus, whatever performance difference exists between MAX and MDMP must come from requests with small stretch values. We refer to Fig. 9 with an intention to observe any such difference.

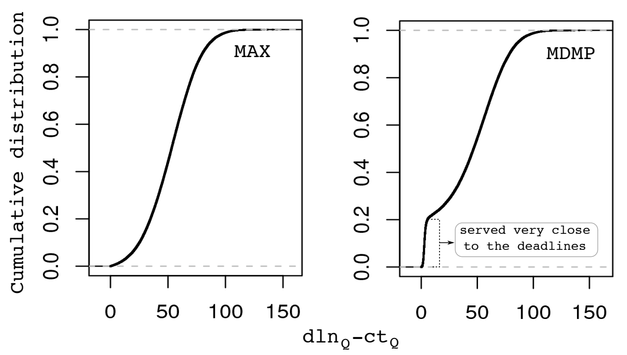

Fig. 9 Cumulative probability distribution of $dln_Q - ct_Q$ (positive slack) for requests which are served within their deadlines ($dln_Q > ct_Q$); $UF_{bs} = 1.0$.

The fundamental difference becomes clear from Fig. 9. The plots show the distribution of slack values (time difference between the completion time of a request and its deadline) for requests that met their deadlines, i.e. $dln_Q > ct_Q$. It seems the poor performance of MAX is attributable to its inability to utilize the time gaps between a request's arrival and its imposed deadline to serve other pending requests. 20% of the requests in MDMP are served very close to their deadlines (near zero positive slack). This indicates that MDMP manages to utilize the slack between a request's arrival and its deadline in serving more urgent requests. Requests far away from their deadlines will have higher probabilities of meeting their deadlines and hence can be pushed back in the schedule to accommodate requests with lower probabilities of meeting their deadlines. The hypothetical deadline created by MAX cannot capture this logic since it attempts to service a request as close as possible to its service time. An important conclusion from this analysis is that the stretch of a request, at least in its defined form, is not an accurate factor to include in deciding scheduling policies. This is particularly true for deadline based systems where using any introduced slack may be advantageous.

7.7 Broadcast Policies

The final analysis presented in this paper is on the broadcast policies generated by heuristics that result in additional performance improvement. For this we observe the policies generated by MBSP and NPRDS. The performance difference between the two heuristics diminish as the utilization factor increases, with NPRDS gaining at very high utilization. The policies are observed in terms of the request and broadcast frequency of data items. To do so, we divided the data items into four access types: *frequent* (ranks 1 to 20), *above average* (ranks 21 to 45), *average* (ranks 46 to 255) and *infrequent* (ranks 256 to 400). We then observed how the broadcast frequency of data items in the four categories differ across the two heuristics. Factors such as request accumulation, broadcast time and request rank appear as major aspects underlying the differences in the broadcast policies.

Fig. 10 illustrates the broadcast frequencies of the data items according to the four access categories. The better performing heuristic is marked with a dotted line in the plots. Table 2 summarizes the broadcast frequencies of the better performing heuristic relative to the other.

Access type	$UF_{bs} = 0.75$	$UF_{bs} = 4.0$
frequent	Lower	Higher
above average	Lower	Lower
average	Similar	Higher
infrequent	Higher	Lower

Table 2 Relative broadcast frequencies of data items according to access types. Each entry signifies if the frequency in the better performing heuristic is lower, similar, or higher than that in the other heuristic.

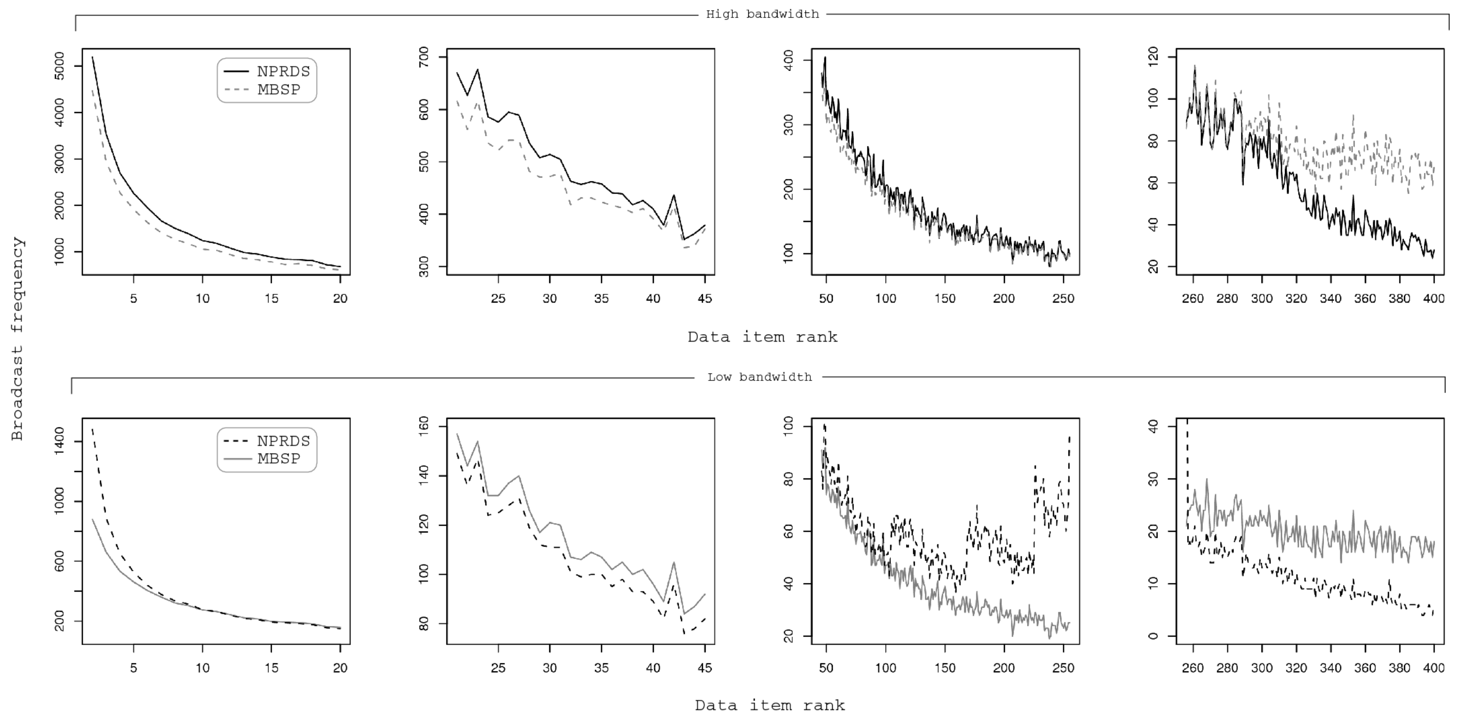

Fig. 10 Broadcast frequency of different data items by their ranks using MBSP and NPRDS. Top: Low utilization ($UF_{bs} = 0.75$). Bottom: High utilization ($UF_{bs} = 4.0$). The dotted lines signify the better performing heuristic.

In a high bandwidth (or low utilization) situation, it may be reasonable to allow more accumulation of frequent data items. This is not only because more requests can be served by a single broadcast, but also because it helps assign more broadcast time to infrequent items. Assuming that access frequencies for data items are the same across different workload conditions, there will not be many requests for infrequent items in a low workload. Hence, a scheduler should not wait for request accumulation of infrequent items and should broadcast the items on a regular basis. It can happen that such a broadcast serves a single request only. Regular broadcasts of infrequent items (larger sizes in our data set) do not induce much delay for the other requests owing to the high bandwidth available. The broadcast frequencies of MBSP is in conformation to this intuition.

The broadcast policy is quite different when the bandwidth available is low, or there is heavy utilization. The plots clearly show differences in the broadcast frequencies of average and infrequent items. Observe that NPRDS has higher broadcast frequencies for frequent and average data items. Higher broadcast patterns for frequent items is crucial in low bandwidths since the broadcast times of items will be typically high. Allowing too many such requests to accumulate and then broadcasting over a slow channel can introduce deadline misses for a higher number of requests. On the other hand, accumulation of average items should not be allowed since it would usually take a long time for a significant accumulation to happen. This will only keep such requests pending in the system for a long duration of time. The request accumulation of above average items are somewhere in between frequent and average ones. NPRDS decides to allow the accumulation of these requests more than MBSP to utilize the broadcast bandwidth for frequent and average items. Infrequent items are ignored the most owing to their high broadcast time.

Understanding the broadcast policies in this manner not only reveals what factors are more important in different utilization, but also paves the way to transform a pull-based system to a push-based one with similar performance. However, in addition to relative measurements, such conversions will also require the estimation of absolute frequency in a probabilistic manner.

8 Conclusion

In this paper, we introduce the problem of stochastic scheduling in broadcast systems where data items must be retrieved from different data servers prior to broadcasts. We modeled the data servers as $M/G/1-PS$ systems where the available bandwidth is equally divided among pending requests. Schedules are then generated at the broadcast server to serve requests within their imposed deadlines. Two novel heuristics are proposed that use the probability distribution of response times in the data servers to generate schedules. The *Minimum Deadline Meet Probability* (MDMP) heuristic schedules requests based on their probability of meeting the deadline. The *Maximum Bounded Slack Probability* (MBSP) heuristic defines an extended deadline for requests as given by a slack deviation parameter, with an intention to use any negative slack for serving other pending requests. The performance of these heuristics is compared to that of four other heuristics – RxW, MAX, SIN2 and NPRDS – that have been proposed for deterministic broadcast scheduling. Performance is evaluated using three metrics - deadline miss rate, average stretch, and utility.

Empirical results on a synthetic data set reveals that both MDMP and MBSP show superior performance in conditions of both low and high broadcast utilization. In very heavy workloads, the response time of data retrieval is observed to become negligible compared to the broadcast time. NPRDS provides a better performance under such situations. However, such load conditions are often not realistic. Further analysis on the impact of various factors on broadcasting policies reveal that size of data items can often be ignored in scheduling decisions when the broadcast bandwidth is high. The stretch of requests, as typically defined, is also found to be an inaccurate factor to consider while scheduling in deadline based systems. Finally, good broadcast policies can have very different characteristics depending on workload conditions. Request accumulation, data item access frequencies and their broadcast times assume different levels of prominence in different situations, thereby affecting the performance of the heuristics. We also explored the effect of the slack deviation parameter on MBSP and found correspondence of its performance to the amount of negative slack that the heuristic is allowed to exploit. Typically, very low values do not enable any slack utilization, while high values provide too much of it to make the broadcast of larger data items feasible.

In future, we plan to explore the scenario when multiple servers can host the same data item. The scheduler then has to build a fetching schedule based on perceived response times and evaluate its effectiveness in combination with the broadcast schedule. Interactions between the broadcast and data server can be made more complex by including concepts such as parallel/pre-fetching and data replication. In general, parallel/pre-fetching will change the probabilities of retrieval times of the data items. The final scheduling approach can therefore remain the same. Data replication will have a deeper impact since the probabilities of fetching the same item within a specific period will be different for different data servers. Increasing the number of data servers and distributing frequently accessed data items across them can also provide lower data retrieval times and can correspondingly impact heuristic performance. These issues require a much extensive analysis and thus forms the basis for our future work. The dynamic management of scheduling algorithms is another possible direction. The motivation for this comes from the observation that different heuristics result in broadcast policies having different characteristics, often advantageous in handling changes in the state of the broadcast system.

References

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

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

Aksoy D, Franklin M, Zdonik S (2001), Data Staging for On-Demand Broadcast. In: Proceedings of the 27th International Conference on Very Large Data Bases, pp. 571–580

Barford P, Crovella M (1998), Generating Representative Web Workloads for Network and Server Performance Evaluation. ACM SIGMETRICS Performance Evaluation Review **26**(1), 151–160

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

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

Dewri R, Ray I, Ray I, Whitley D (2008), Optimizing On-Demand Data Broadcast Scheduling in Pervasive Environments. In: Proceedings of the 11th International Conference on Extending Database Technology, pp. 559–569

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

Jensen E, Locke C, Tokuda H (1985), A Time Driven Scheduling Model for Real-Time Operating Systems. In: Proceedings of the Sixth IEEE Real-Time Systems Symposium, pp. 112–122

Lee VC, Wu X, Ng JKY (2006), Scheduling Real-Time Requests in On-Demand Data Broadcast Environments. Real-Time Systems **34**(2), 83–99

Dempster MAH, Lenstra JK, Kan, AHGR (1982) Deterministic and Stochastic Scheduling. D. Reidel Publishing Company

Megow N, Uetz M, Vredeveld T (2006), Models and Algorithms for Stochastic Online Scheduling. Mathematics of Operations Research **31**(3), 513–525

Omotayo A, Hammad MA, Barker K (2006), Update-Aware Scheduling Algorithms for Hierarchical Data Dissemination Systems. In: Proceedings of the 7th International Conference on Mobile Data Management, p. 18

Press WH, Vetterling WT, Teukolsky SA, Flannery BP (1992) Numerical Recipes in C: The Art of Scientific Computing Second edn. Cambridge University Press, pp. 538–545

Ravindran B, Jensen ED, Li P (2005), 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, pp. 55–60

Su CJ, Tassiulas L (1997), Broadcast Scheduling for Information Distribution. In: Proceedings of the INFOCOM '97, pp. 109–117

Sun W, Shi W, Shi B, Yu Y (2003), A Cost-Efficient Scheduling Algorithm of On-Demand Broadcasts. Wireless Networks **9**(3), 239–247

Triantafillou P, Harpantidou R, Paterakis M (2002), High Performance Data Broadcasting Systems. Mobile Networks and Applications **7**(4), 279–290

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

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