Demand Bound Server: Generalized Resource Reservation for Hard Real-Time Systems

Pratyush Kumar
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
kumarpr@tik.ee.ethz.ch

Jian-Jia Chen
Department of Informatics
Karlsruhe Institute of Technology (KIT), Germany
jian-jia.chen@kit.edu

Lothar Thiele
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
thiele@tik.ee.ethz.ch

ABSTRACT

Servers have been proposed to implement resource reservations on shared resources. Such reservations isolate the temporal behavior of tasks sharing the shared resources, thereby providing performance guarantees to tasks independent of other tasks. In existing work, resource reservation has been synonymous to utilization (also called bandwidth) on the resource, i.e., we can reserve only a constant fraction of the resource utilization via a server. Such reservation schemes are not suited to serve interrupt-like tasks: tasks that occur seldom but require quick service or tasks with jitter. With this motivation, we present a generalized server algorithm, called Demand Bound Server (DBS), whose offered service is characterized by the demand bound function (`dbf`) of the task it serves. We show that schedulability of DBS tightly follows that of EDF, and if schedulable a DBS provides a performance guarantee as requested by the `dbf` of the task. We present an implementation of DBS when the `dbf` is a shifted-periodic curve and characterize its overhead. We also present efficient composition operations on DBS that widen the class of implemented servers to tightly serve tasks arising in most practical settings.

Categories and Subject Descriptors

D.4.7 [**Software**]: Operating systems—*Real-time systems and embedded systems*

General Terms

Design, Performance, Theory

Keywords

Real-Time Systems, Servers, Timing Isolation, Composition

1. INTRODUCTION

Often, multiple tasks share the same computing resource. As a result, the timing behavior of these tasks influence each other due to interference on the shared resource. In many systems, in particular hard real-time systems, it is important to limit the interference between tasks in order to provide performance guarantees to tasks independently upon the working of other tasks. Such independent guarantees allow for a composable design approach, for instance tasks with different degrees of criticality can coexist. Servers have been proposed to implement such resource reservations. They provide means to virtualize or partition the available computing resource in terms of timing properties.

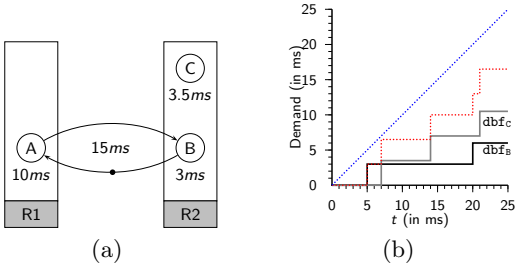


Figure 1: (a) System considered for first example, and (b) EDF schedulability analysis on R2.

Traditionally, the quantum of reservation has always been a fixed fraction of the resource, called utilization (or bandwidth). For instance, with dynamic priority servers such as the two-level hierarchical scheduler in [1], Dynamic Priority Exchange Server (DPE) [2], the Dynamic Sporadic Server (DSS) [3, 4], the Total Bandwidth Server (TBS) [2], the Constant Bandwidth Server (CBS) [5], the Bandwidth Sharing Server (BSS) [6] and the PShED algorithm [7] we can reserve a specific utilization on the resource, usually denoted as U_s. It is important to note that even with arbitrary hierarchical composition of such servers, the unit of resource reservation remains a fraction of the resource.

Such an approach has some clear advantages. Firstly, proving that multiple servers can co-exist in a system, i.e., proving schedulability, amounts to simply adding up the utilization of all servers and checking if it is below the total available system utilization. This simple interface between the server and the system leads to the second advantage, namely, a reserved fraction of the resource can be hierarchically divided into other such fractions which can then be utilized using different scheduling policies.

However, there is a disadvantage of reserving only fractions of the resource. We discuss this with some examples. Consider two resources R1 and R2 scheduling three tasks A,

EMSOFT'11, October 9–14, 2011, Taipei, Taiwan.

B and C, (Figure 1(a)). Task A is mapped onto R1, while tasks B and C are mapped onto R2. Tasks A and B belong to a dataflow graph that executes periodically with period 15ms. The execution times of tasks A and B are 10 and 3 ms, respectively. It is required that one iteration of the dataflow graph complete before the next iteration begins. Task C is an independent periodic task with execution time 3.5ms, and period and relative deadline equal to 7ms. To meet the performance constraint of the dataflow graph, task B must complete within 5ms of its invocation. We need to reserve at least $3/5 = 60\%$ ($= U_B$, say) of R2 to serve task B. This reservation may be implemented by any of the server algorithms mentioned above. To serve C, while satisfying its performance requirement, we need to reserve at least $3.5/7 = 50\%$ ($= U_C$, say) of R2. As $U_B + U_C = 110\% > 100\%$, the two servers, cannot be co-scheduled on R2 while satisfying the performance guarantees of the tasks.

Instead of using servers, let B and C execute on R2 using the EDF scheduling policy, with relative deadlines of 5ms and 7ms, respectively. Such a system is indeed schedulable and performance guarantees of tasks are met. This is shown in Figure 1(b): the sum of the demand bound functions of the two tasks (shown in red), is below the line through the origin and with slope 1 (shown in blue). This illustrates that isolating tasks with servers can sometimes come at a hefty price, that of in-feasibility. In more moderate situations, it may come at a price of unnecessary over-provisioning.

Apart from the case of actors of a dataflow graph, the limitation of existing servers can be noticed in other situations as well. We briefly describe two other such situations. Firstly, consider the case of interrupt service routines (ISRs) which usually perform low-level device management functions or execute kernel operations. It is essential that they execute almost *immediately*, i.e., their relative deadline is close to their execution time. If we serve such tasks via a server, then the server must reserve the full resource, providing no opportunity to compute performance guarantees of other tasks. However, if the frequency of the ISRs is bounded as proposed in [8], it is indeed possible to schedule, with performance guarantees, other tasks. A similar effect is seen in the case of tasks with bursts such as periodic tasks with jitter. A server serving such a task must reserve the utilization to accommodate the burst, independent of the long-term behavior of the task. It is possible that such over-provisioning leads to non-schedulable systems, even though the system is EDF-schedulable.

There are two points which may be noted. The first point is related to resource reclamation with servers. Consider a different reservation scheme for the first example: Let B be served by some server with its desired utilization of 60%, and let C be served by another server with the remaining available utilization of 100 - 60 = 40%, such that the overall system is schedulable. If the server serving C allows for resource reclamation, as proposed in [9], [10] or [11], then C would indeed complete within its deadline, as its server would scavenge unused utilization from the infrequently activated task B. However, this approach does not achieve what we set out to do, namely, to isolate tasks with servers such that performance guarantees of tasks can be met *independent of other tasks*. The interface of utilization used in existing servers does not suffice to tightly follow the EDF schedulability constraint.

The second related point is about interfaces, in the design and analysis of real-time systems. Certainly, interfaces exist that greatly generalize the utilization interface. Well known examples include the periodic supply function proposed in [12] and resource curves used in network [13] and real-time calculus [14]. However, the key point is whether there are server algorithms which guarantee such general interfaces of resource usage, and thereby implement task isolation. To the best of our knowledge, only utilization-based interfaces have been considered for resource reservations.

Thus, while the utilization-based reservation schemes have their advantages, they suffer from the disadvantage of having a *schedulability gap* to the optimal EDF schedulability constraint, i.e., there are EDF-schedulable task-sets which are not schedulable under resource reservation using servers. The goal of this paper is to define and formulate a new class of servers, called Demand Bound Servers (DBS), that is able to bridge this gap. DBS provides tasks it serves a service guarantee that is described as a demand bound function, while implementing task isolation. The schedulability test of DBS is the same as demand bound test for EDF. From this tightness we obtain the following optimality property: Whenever a task-set is schedulable by a dynamic priority server, it is schedulable by some configuration of DBS.

The proposed demand bound server generalizes the reservation guaranteed to a task independently upon other tasks. However, as expected, this generalization comes at a price. Instead of the simple utilization, the interface between the demand bound server and the system is given by the demand bound function. Consequently, the schedulability analysis and the server algorithm are more involved. One may argue that the schedulability analysis is offline and thus attention may be limited to efficient server algorithms. To this end, we take two steps. Firstly, we propose the Shifted-Periodic Demand Bound Server (SP-DBS), which is a DBS that guarantees a shifted-periodic demand bound function and has an efficient implementation. Secondly, we propose composition operations on DBS which can hierarchically combine several DBS into one. This enables us to extend the class of SP-DBS to a much wider class of DBS.

The results of the paper can be summarized as follows:

- We define a new class of dynamic priority servers, named Demand Bound Server (DBS), which closes the discussed schedulability gap.
- We provide the schedulability test and the performance guarantee of the server, such that real-time analysis of general streams of jobs can be analyzed. To the best of our knowledge, such analysis of hard real-time properties has not been presented even for existing servers.
- We provide a provably correct implementation of the demand bound server if it guarantees a periodic service. We also characterize the overhead of such an implementation.
- We present composition operations to efficiently implement a richer class of DBS.

The rest of the paper is organized as follows. In Section 2, we define a Demand Bound Server (DBS) and formulate its defining properties. In Section 3, we present the implementation of the Shifted-Periodic Demand Bound Server and prove that it is indeed a DBS. In Section 4, we discuss compositional operations to implement a richer class of DBS, and finally conclude in Section 5.

2. DEMAND BOUND SERVER

In this section, we present the definition of a Demand Bound Server (DBS). We also present defining properties that such a server must provide. We show how these properties can be used to verify schedulability and to compute the worst-case response time of tasks served by a DBS.

Note that in this paper, we do not consider the overhead due to context switching or task scheduling. Approaches that have been used in case of other servers, like CBS, could be applied as well in order to estimate the corresponding influence, but this is out of scope for the present paper.

2.1 System model

We start by describing our task model and deriving some useful quantities. A real-time task τ can be considered as a stream of jobs. Each job J_j is characterized by an arrival time a_j, an absolute deadline d_j, and an upper bound on its execution time C_j. As a server executes jobs in First-Come-First-Serve (FCFS) order, we suppose that the absolute deadlines of jobs are non-decreasing in the order of arrival. There is no restriction on the arrival times of jobs such as periodicity or minimal inter-arrival time.

We define the input trace R of a task as follows:

$$R(t) = \sum_{\{j \mid a_j < t\}} C_j. \tag{1}$$

The input trace, R, specifies a bound on the cumulative resource demanded by all jobs that have arrived until a given time. Note that in the above equation the summation is over all a_j strictly less than t. Thus, if a job arrives at say $a_j = 2$, then its resource demand is not accounted for in $R(2)$ but in $R(2+\epsilon)$, for any $\epsilon > 0$. Alternately stated, R is a left-continuous function.

In order to define the resource demand of a general sequence of jobs, we use the demand bound function (DBF) that was first proposed by Baruah *et al* in [15].

Definition 1 (Demand Bound Function). *A task has a demand bound function,* $\texttt{dbf}$*, if for any* $\Delta > 0$*, in an interval of time* $[t, t+\Delta]$ *for any* $t > 0$*, the sum of the execution times of all jobs that arrive not earlier than* t *and have deadline not later that* $t + \Delta$*, does not exceed* $\texttt{dbf}(\Delta)$*.*

Since the DBF is the upper-bound on the resource requirement of a task, it can be used to validate the schedulability of a system, see e.g. [15]. For instance, if tasks τ_i with DBF $\texttt{dbf}_i$ share a computation resource under EDF scheduling, they are schedulable iff

$$\sum_i \texttt{dbf}_i(t) \leq t, \quad \forall t \geq 0. \tag{2}$$

To provide timing isolation, it is required that we serve the tasks via servers. If each task is to be insulated for every other task, each task may be served by a different server. It may also happen that a certain group of tasks are related and there is no need for temporal isolation amongst them, for instance if the tasks belong to the same application. We may serve such a group of tasks within a single server. Within the demand bound function modelling framework, conceptually, we may consider the jobs from such a group of tasks as belonging to a single task. This single task should have a demand bound function given as the sum of the demand bound functions of the individual tasks of the group. As can be seen such an interpretation retains the tightness to the schedulability condition of (2). Thus, without loss of generality, we may always consider that we have tasks, each of which has to be served by a different server. The tasks here already reflect the desired boundaries of temporal isolation.

2.2 Defining the server

A server must be characterized by two guarantees. Firstly, a guarantee must be provided on the maximum amount of resource that the server can consume. Such a guarantee is needed to analyze the schedulability of a set of servers and/or tasks executing on a resource. Secondly, the server must provide a service guarantee to the task that it serves. Such a guarantee can be used to compute the worst-case response time (WCRT) of jobs served by the server.

In this work, we define the Demand Bound Server (DBS) as a dynamic priority server that is characterized by a demand bound function denoted as $\texttt{dbf}_s$. Both the guarantees that a DBS has to provide will be given in terms of its defining demand bound function $\texttt{dbf}_s$.

We first define some notation that characterizes the working of a DBS. To serve the stream of jobs, the DBS must make a sequence of requests for execution on the resource. Since the underlying scheduling is EDF, each such request is characterized by a 3-tuple (x, y, z) interpreted as follows: At time x the server requests z amount of resource to be provided within an absolute deadline y. Let ρ denote the set of all such 3-tuples that denote the requests made by the DBS to the underlying EDF scheduler.

Having established the notations, we now present the guarantee of the maximum resource consumed, as the first defining property of a DBS.

Definition 2 (DBS Property I). *A DBS characterized by the demand bound function* $\texttt{dbf}_s$ *when serving a stream of jobs must satisfy the following property:*

$$\sum_{\{(x,y,z)\in\rho \mid x \geq t \wedge y \leq t+\Delta\}} z \leq \texttt{dbf}_s(\Delta), \qquad \forall\ t,\ \Delta \geq 0. \tag{3}$$

Indeed, the above property is similar to the definition of a demand bound function. It says that a DBS shall not request for more resources than a task with a demand bound function $\texttt{dbf}_s$. Thus, we can use the standard EDF schedulability test of (2) to verify the schedulability of a DBS when executed together with other servers and/or tasks. Noted that because of the underlying EDF scheduling discipline, a DBS can be co-scheduled along with other dynamic priority servers such as CBS or with other tasks executing directly on the resource. In other words, the DBS inherits the underlying compositionality of the EDF resource model.

We now present the second property that provides a service guarantee to tasks. To this end, we first define the output trace, denoted as $R'(t)$, as the accumulated amount of execution time provided to jobs served by the server until time t.

Definition 3 (DBS Property II). *Let a stream of jobs with an input trace* R *be served by a DBS characterized by* $\texttt{dbf}_s$*. Let the resultant output trace be given by* R'*. Then, for any* $t \geq 0$ *there exists* s*,* $0 \leq s \leq t$*, such that*

$$R'(t) \geq R(s) + \texttt{dbf}_s(t - s). \tag{4}$$

Firstly, note that $(R'(t) - R(s))$ denotes the total amount of resource consumed by all jobs that arrived after s and completed execution on or before t. Given this interpretation, the above property says that for *any* time t, there is *some* interval $[s, t]$ such that the jobs that have arrived after s receive at least $\texttt{dbf}_s(t-s)$ service no later than t. This is the guarantee that the server provides to the jobs it executes.

It is important to distinguish between the above guarantee and similar concepts of a lower service curve in Real-Time Calculus (RTC) [14] and supply bound function defined in [12]. All of these concepts are designed to enable computation of response times of tasks. The lower service curve, β^l, as used in RTC, guarantees that in *every* interval of any length Δ, the minimum amount of execution guaranteed is $\beta^l(\Delta)$. Supply bound function is similarly defined. DBS Property II, on the other hand, provides no such guarantees, for every interval. Similar to network calculus described in [13], it states that for every time instant, there exists *some* interval, ending at this time instant, for which (4) holds.

2.3 Analysis of the server

Having presented the defining properties of the DBS we now discuss the use of these properties to analyze a DBS. We present the results in the subsequent theorems.

THEOREM 1 (SCHEDULABILITY OF DBS). *Let a set $S = \{DBS_i\}$ denote a set of DBSs executing on a resource, such that DBS_i is characterized by a demand bound function $(\texttt{dbf}_s)_i$. Then, the set of servers is EDF schedulable iff*

$$\sum_{\{i \mid DBS_i \in S\}} (\texttt{dbf}_s)_i(t) \leq t, \quad \forall t \geq 0. \tag{5}$$

All proofs are presented in the appendix.

We now turn our attention to computing the WCRT of a stream of jobs. Since the jobs execute in FCFS order, the WCRT is simply the maximal horizontal distance between the input trace R and the output trace R', denoted as $\texttt{Del}(R, R')$. This distance can be mathematically formulated as

$$\texttt{Del}(R, R') = \sup_{t \geq 0} \left\{ \inf\{\Delta \geq 0 : \ R(t-\Delta) \leq (R')(t)\} \right\}. \tag{6}$$

Using this definition we have the following theorem.

THEOREM 2 (PERFORMANCE GUARANTEE OF DBS). *When a stream of jobs with an input trace R is served by a DBS characterized by $\texttt{dbf}_s$, we have*

$$WCRT \leq \texttt{Del}(R, R \otimes \texttt{dbf}_s), \tag{7}$$

where,

$$(f \otimes g)(t) \stackrel{def}{=} \inf_{0 \leq \lambda \leq t} \{f(t-\lambda) + g(\lambda)\}. \tag{8}$$

The above result prescribes how to compute the WCRT for a given stream of jobs. This can be generalized further. Using principles of network and real-time calculus, one can describe a class of possible input traces by an arrival curve $\alpha(\Delta)$. It bounds the maximal accumulated execution time of tasks arriving in any time interval of length Δ, i.e.,

$$R(t) - R(s) \leq \alpha(t-s), \forall s < t. \tag{9}$$

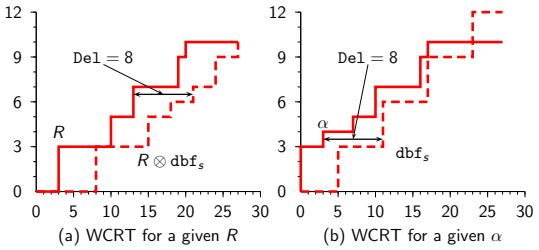


Figure 2: Example illustrateing WCRT computation

Known results as described in [13] show that the worst-case response time of jobs can also be bounded by the maximal horizontal distance between the functions $\alpha(t)$ and $\texttt{dbf}_s(t)$, i.e. WCRT $\leq \texttt{Del}(\alpha, \texttt{dbf}_s)$.

To illustrate the computation of the WCRT, we use an example from a later section. Consider a DBS, with a shifted periodic $\texttt{dbf}_s$ as shown in Figure 2(b). When an input R as shown in Figure 2(a), is served by this DBS, the WCRT is computed as the maximal horizontal distance between R and $R \otimes \texttt{dbf}_s$. For the given input trace, R, we compute the arrival curve α which is shown in Figure 2(b). The WCRT for any input trace conforming to this arrival curve is obtained by the maximal horizontal distance between α and $\texttt{dbf}_s$.

Recall that in Section 1 we motivated the definition of the DBS by the schedulability gap that we noticed in the use of other dynamic priority servers. We are now ready to show that a DBS characterized by DBS properties I and II defined in Definitions 2 and 3, can bridge this schedulability gap.

THEOREM 3 (TIGHTNESS OF DBS). *If a set of tasks, each characterized by a demand bound function is EDF schedulable, then if we serve each task by a DBS characterized by the demand bound function of that task, the servers are schedulable and real-time requirements of all tasks are met.*

In the above theorem we use the notion of task as discussed in Section 2.1: A group of tasks can be represented by a single task with demand bound function equal to the sum of the demand bound functions of the individual tasks.

From the above result it follows that the DBS follows the EDF schedulability constraint in addition to providing timing isolation. From this tightness result and the known tightness of the EDF schedulability test, we can derive the following optimality property of DBS: *For any given set of tasks with associated demand $\texttt{dbf}_\tau$, if there exist no demand bound servers that can serve these tasks, then there exist no other dynamic priority servers that are able to.*

In addition, it follows from the above theorem, that a DBS characterized by $\texttt{dbf}_s$ that satisfies $\texttt{dbf}_s(t) \geq \texttt{dbf}_\tau(t)$ for all t also guarantees the required demand $\texttt{dbf}_\tau$. In other words, one could approximate a potentially complex demand bound function $\texttt{dbf}_\tau$ by a server function $\texttt{dbf}_s$ that allows for a simple and efficient implementation. Indeed, such an approximation will lead to loss of tightness to the optimal EDF schedule.

3. SHIFTED PERIODIC DEMAND BOUND SERVER

So far, we have defined and characterized a Demand Bound Server (DBS). While the DBS is able to bridge the schedu-

lability gap, it is a very generic model of a server. In particular, we are interested in specific classes of DBS with an efficient implementation. To this end, we devote the remainder of this paper.

In this section, we present the implementation of a specific kind of a DBS where the demand bound function $\texttt{dbf}_s$ is a periodic stair-case function with arbitrary initial offset. Such a function formally specified in (10) will be simply referred to as shifted-periodic, and the server will be called Shifted Periodic Demand Bound Server (SP-DBS). Its $\texttt{dbf}_s$ is characterized by 3 parameters: a maximum capacity Q_s, a period P_s and a deadline D_s. In the remainder of the paper, we say that a SP-DBS is characterized by the tuple (Q_s, P_s, D_s). In terms of these parameters, the $\texttt{dbf}_s$ of a SP-DBS is given as

$$\texttt{dbf}_s(\Delta) = \max\left\{0, \left(\left\lfloor \frac{\Delta - D_s}{P_s} \right\rfloor + 1\right) \times Q_s\right\}. \qquad (10)$$

3.1 General representation of dynamic priority servers

Before we describe our implementation of a SP-DBS, we first formulate a generic representation of dynamic priority servers. An EDF server has three components: a server, its associated task queue and an EDF scheduler. The behavior of the system is governed by signals passed between the components. The EDF scheduler is common across all EDF servers on that resource.

EDF scheduler: The EDF scheduler implements the standard EDF policy, but with two minor changes. Firstly, the EDF queue contains pointers to task queues instead of pointers to individual jobs. The EDF scheduler assigns the processing resource to the task queue with smallest deadline and keeps track of the amount of execution it receives in a variable δ. Secondly, if either the task queue becomes empty or if δ equals the capacity of the task queue, the EDF scheduler removes the pointer to the task queue from its EDF queue, generates a signal `TQRemoved`, returns the value of δ to the server algorithm and then resets δ to 0.

Task queue: The task queue contains the arrived and not finished jobs of the task that is served via the server. To each task queue there is associated an absolute deadline d that denotes its dynamic priority, a non-negative capacity c, and a re-insertion time r that denotes when the task queue must be re-inserted into the EDF queue, if it is not already in the EDF queue. When the task queue is empty and a new job arrives, the task queue generates a signal `TQNonEmpty`.

Server: The server algorithm is called when either of the two above signals, `TQNonEmpty` and `TQRemoved`, are generated. The server algorithm is responsible for updating the task queue parameters d, c and r.

With such a model of the system, describing different dynamic priority servers amounts to describing the initial conditions and what the server algorithm does in response to the two signals `TQNonEmpty` and `TQRemoved`.

3.2 Implementation of SP-DBS

We now discuss an implementation of a SP-DBS. At each instant a SP-DBS maintains three server variables, namely, t', c' and a replenishment set η. The variable t' denotes the time of the last request for resource by the server and c' the task queue capacity at time t'. The replenishment set η is composed of a set of tuples (u, v), which denotes that at time u a replenishment by amount v can be performed.

3.2.1 Server Algorithm

In the description below, let t denote the current time.

1. Initial conditions
   (a) Initialize server variables: $c' := Q_s, t' := 0, \eta := \phi$.
   (b) Initialize task queue parameters: $c := Q_s, d := 0, r := \infty$.
2. When `TQNonEmpty` is generated
   (a) The deadline of the task queue is changed to $d := \max\{d, t + D_s\}$.
   (b) We set $t' := d - D_s$.
   (c) $r := d - D_s$.
3. When `TQRemoved` is generated
   (a) $c := c - \delta$.
   (b) An entry $(t' + P_s, c' - c)$ is inserted into η.
   (c) If $c = 0$
   For all entries, (u, v) with $u \leq t$, we increase the value of c by v and delete that entry.
   If there are no such entries we pick the oldest entry in (u, v) set the deadline $d := \max\{d, u + D_s\}$ and the capacity $c := v$. We delete that entry from η.
   (d) We set $c' := c$.
   (e) If the task queue is not empty, we set $r := d - D_s$ and $t' := d - D_s$.

Note that for an implementation, the set η can be maintained as a FCFS queue. Entries (u, v) are added in Step 3(a) with increasing values of u. Entries (u, v) with the smallest values of u are removed in Step 3(b). Thus, both inserting and deleting entries from η are constant-time operations. We discuss the complexity in greater detail in Section 3.4.

3.2.2 Example

Consider two SP-DBSs: DBS_1 and DBS_2 with properties $(3, 6, 5)$ and $(1, 3, 2)$, respectively. It can be verified that these two SP-DBSs are EDF schedulable. We illustrate how these two servers work on a resource when serving an input trace that is shown in Figure 3. The arrival of tasks is shown with upward arrows with the execution time of the task (unknown to the server) depicted on top of the arrow. In the figure, the execution pattern, the deadline d and the capacity c of both the servers are plotted over time. Dark boxes under the deadlines denote times the task queue in the EDF queue. Also plotted are R, R' and $R \otimes \texttt{dbf}_s$. Note that for both the servers, $R' \geq R \otimes \texttt{dbf}_s$.

3.3 Analysis of SP-DBS

With the subsequent results we establish that SP-DBS with its proposed implementation is indeed a DBS.

THEOREM 4. *SP-DBS satisfies DBS Property I.*

THEOREM 5. *SP-DBS satisfies DBS Property II.*

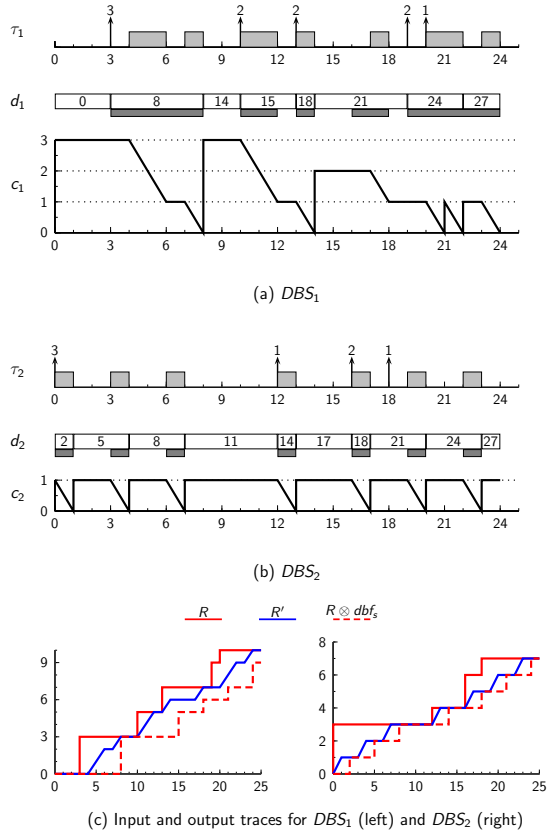


Figure 3: Example of two SP-DBSs

The DBS property I of SP-DBS can be checked in the example trace of Figure 3.

To understand DBS property II of SP-DBS better, let us discuss the computation of s, that satisfies (4), for certain times t for the example shown in Figure 3. First consider, $t = 7$ in DBS_2. At this time, the deadline is 8, and indeed at $s' = 8 - D_s = 6$ there is a request for resource. This request is because of a replenishment performed. We reduce the value of s' by $P_s = 3$, two times, to obtain $s' = 0$. At $s' = 0$, job 1 of task τ_2 arrives while the server is idle, and hence, we set $s = s' = 0$. Indeed, in $[0, 7]$ the DBF Property II holds: $R'(7) - R(0) = 3 = \texttt{dbf}_s(7)$.

Now consider $t = 8$ for DBS_1. We have deadline $d = t = 8$. So, this belongs to Case (b) of the proof above. Indeed in the interval $[3, 8]$, we have a tight schedulability constraint: the sum of the requests by the two servers equals 5. We set $s = 3$ and have the desired result.

Finally, consider time $t = 18$, for DBS_1. The third job of task τ_1 arrives when the task queue is empty at time 13, and takes $5 (= D_s)$ units of time to complete $2 (< Q_s = 3)$ units of execution. Prima facie, this appears to violate the resource guarantee of the server. Let us apply the argument of the above proof. At $t = 18$, the deadline is $d = 21$. Indeed, at $s' = d - D_s = 16$, we had a replenishment. By reducing $s' = 16 - P_s = 10$, we arrive at another request when a job arrived while the task queue was empty. So, we set $s = 10$. Indeed, in the interval $[10, 18]$ the DBF property II holds on DBS_1.

Let us revisit the first example discussed in the Section 1. Existing servers are not able to schedule the tasks. As shown in Figure 1 the `dbf`s of tasks B and C are shifted periodic curves. They can, thus, be served by SP-DBS with the `dbf` of the task equal to the server `dbf`. From the results in this section, such a system is schedulable and real-time constraints of the tasks are met.

3.4 Implementation complexity

From Theorem 3, we know that if a set of tasks have shifted periodic demand bound functions, and are schedulable by EDF, then a configuration of SP-DBS exists that can serve them. By implementing the server on top of the EDF scheduler, we provide task isolation. But this advantage is obtained with the additional overhead of the server algorithm. To characterize this overhead we analyze the worst case memory and timing complexity of the server algorithm.

Memory complexity.

We ignore the constant memory overhead of maintaining server variables such as c' and t'. We focus on the replenishment set η.

To characterize the memory complexity, we define a parameter of the task set c_{min}. Let c_{min} be the smallest number such that the execution times of all tasks and the parameter Q_s are integral multiples of c_{min}. We can consider c_{min} as the least unit of the capacity c and thus, the smallest amount of resource requested by the server, at any time, is at least c_{min}. Then, for all entries (u, v) in η, we have $v \geq c_{min}$. An entry (u, v) in η indicates that a replenishment by amount v can be performed at time u. Such an entry is created (in Step 3(a)) if a resource v was requested by the server at time $u - P_s$. Thus, the used capacity is replenished by the same amount at a later time. This means that the capacity never exceeds the initial capacity of Q_s. In Step 3(b), we delete all entries of (u, v) in η with $u \leq t$ and add v to c. If the capacity c does not fall to 0, for a sufficiently long time, we will accumulate the v of all existing entries of η in c. still have a capacity $c \leq Q_s$. Thus, at any point of time, the sum of v across all entries of η does not exceed Q_S. Using this property of η and the defined parameter c_{min}, the number of entries in η, at any point of time, is at most Q_s/c_{min}.

Timing complexity.

As discussed, because of the FIFO queue property of η, the execution of the server algorithm by itself is constant time. However, the server algorithm requests for inserting its task queue into the EDF queue at different points of time in Steps 2(c) and 3(d). The number of insertions are made until time t is at most $R'(t)/c_{min}$. Further, we know that $R'(t) \leq \texttt{dbf}_s(t + D_s) \leq Q_s + (Q_s/P_s)t$. Thus, the maximum number of insertions made to the EDF queue, for any input, grows linearly in time, and is inversely related to the parameter c_{min}.

An ordinary EDF scheduler working on a stream of jobs instead of the task queue, must also perform insertions of individual jobs into the EDF queue. In the worst-case, it can be argued that the number of insertions that the server performs is at most Q_s/c_{min} times the number of insertions made by the original EDF scheduler.

In conclusion both the memory and timing complexity of the server algorithm are bounded and inversely grow as the granularity of the jobs, defined in c_{min}, falls.

3.5 Soft variant of SP-DBS

In the literature, soft[1] variants of dynamic priority servers has been proposed, for example the original Constant Bandwidth Server proposed in [5] is a soft server. In such implementations, whenever the task queue served by the server is non-empty, it is inserted into the EDF queue, i.e., the server is never suspended. Both hard and soft variants have their relative merits and demerits as discussed in [16].

The presented algorithm of SP-DBS is a hard implementation. With the template of a general dynamic priority server presented in Section 4.1, differentiating between soft and hard variants is straight-forward. A soft implementation requires only one change: whenever the task queue reinsertion time r is set, it must be set to the current time t. It can be proved that with this change, the DBS properties I and II still hold.

4. COMPOSITION OPERATIONS ON DBS

Recall that, with suitably dimensioned SP-DBS, we were able to serve tasks of the first example of Section 1, while proving schedulability and providing task isolation. Now let us consider another example with a task with a burst. Task E is a periodic task with period 1.5, a maximum jitter of 0.5 that can be exhibited by a maximum of 3 consecutive events, an execution time of 1 and a relative deadline of 3. Another periodic task F has a period 6, execution time 1.5 and relative deadline 2. As before, existing utilization-based servers cannot serve the two tasks though the system is EDF schedulable. Can SP-DBS serve this task-set? Because the demand bound function of E is a shifted-periodic function it cannot be tightly served by a SP-DBS. However, we can design a SP-DBS S_E with $(\texttt{dbf}_s)_E > \texttt{dbf}_E$ and use it to serve task E. Since, in the long-run, $\texttt{dbf}_E$ is periodic with period 1.5 and height 1, we can serve it with a SP-DBS with parameters (1, 1.5, 1.5). The deadline parameter is set such that, $(\texttt{dbf}_s)_E \geq \texttt{dbf}_E$. Task F, on the other hand, can be tightly served by a SP-DBS S_F with parameters (1.5, 6, 2). The demand bound functions of these two SP-DBS are shown in Figure 5(a). As shown the two SP-DBS are not schedulable together. Indeed, this situation arises because the S_E does not tightly provide the requested `dbf` of task E.

The above example motivates the design of DBS with more general $\texttt{dbf}_s$. Towards this end, we propose to use composition operations on several DBS to form a single DBS. In other words, several DBS, interacting in a specific manner, would be used to serve a single task. The goal is to design operations which have a small implementation overhead while broadening the class of implemented $\texttt{dbf}_s$. In the remainder of this section, we present two such composition operations.

4.1 General template for composition

We first discuss a general template for operations on one or more DBSs. Let an operation on a set of DBS, $\mathbf{S} = \{S_1, S_2, \ldots S_n\}$, compose them to form a single server, S, which serves a specific task queue. Server S provides the *glue logic* to interface the servers in $\mathbf{S}$ with the EDF scheduler, as shown in Figure 4. Let the interface variables c, d, r and δ of server $S_i \in \mathbf{S}$ be denoted as c_i, d_i, r_i and δ_i, respectively. Then, the glue logic of S has the following two

[1]The word "soft" here is not used in the same sense as in soft real-time systems. Soft servers can be characterized by hard real-time properties.

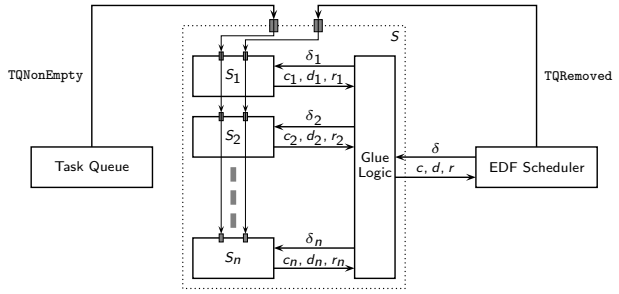


Figure 4: System diagram of composition of servers

roles: (a) compute the task queue parameters c, d and r as a function of the respective parameters of the servers c_i, d_i and r_i of server $S_i \in \mathbf{S}$, and (b) compute the input parameter δ_i for each server $S_i \in \mathbf{S}$ as a function of the input δ from the EDF scheduler. This glue logic is to be executed only whenever the signals `TQNonEmpty` and `TQRemoved` are raised. Also, each of the servers in $S_i \in \mathbf{S}$ has its respective server algorithm which are invoked whenever the signals `TQNonEmpty` and `TQRemoved` are raised.

It must be proved that the operation of the DBS(s) using the glue logic is such that S is indeed a DBS, i.e., S satisfies the DBS properties I and II, when each server S_i in $\mathbf{S}$ satisfies these properties. If so, furthermore, it is desirable that `dbf` that characterizes S be obtained as a function of the `dbf` of the servers in $\mathbf{S}$. Such a composition would prove useful if upon composition, the class of DBS that can be implemented becomes richer. We discuss two such operations.

4.2 Min-composition

We now discuss the *min-composition* operation which is formally defined as follows.

DEFINITION 4 (MIN-COMPOSITION). *A set of DBS $\{S_1, S_2, \ldots S_n\}$ is said to be min-composed to form a server S, denoted as $S = S_1 \wedge S_2 \wedge \ldots \wedge S_n$, if the glue logic of server S implements the following interface*

$$\begin{aligned} d &:= \max(d_1, d_2, \ldots d_n), \\ c &:= \min(c_1, c_2, \ldots, c_n), \\ r &:= \max(r_1, r_2, \ldots, r_n), \\ \delta_i &:= \delta, \qquad \forall i \in \{1, 2, \ldots, n\}. \end{aligned} \tag{11}$$

For the glue logic defined as above, we can prove the following property about the composed server S.

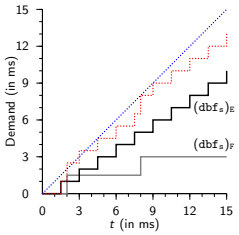
THEOREM 6. *Let $S := S_1 \wedge S_2 \wedge \ldots \wedge S_n$, then the server S is a DBS characterized by demand bound function $\texttt{dbf}_s$ which is related to $(\texttt{dbf}_s)_i$, the demand bound function that characterizes server S_i, $i \in \{1, 2, \ldots, n\}$, as follows*

$$\texttt{dbf}_s := \min\big((\texttt{dbf}_s)_1, (\texttt{dbf}_s)_2, \ldots, (\texttt{dbf}_s)_n\big). \tag{12}$$

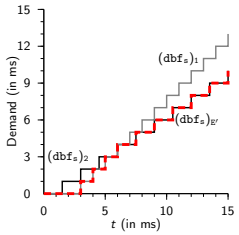
Due to the interface established between servers and the EDF scheduler, such min-composition can be performed hierarchically, thereby improving the efficiency of the glue logic and allowing for a distributed implementation.

4.3 Left-shift

We now discuss a second operation on a single DBS called *left-shift*. The operation is formally defined as follows.

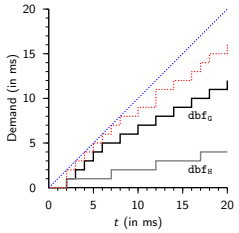


(a) Serving with SP-DBS is not schedulable

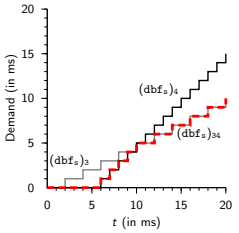


(b) Min-composed $S_{E'}$ has $(\texttt{dbf}_s)_{E'} = \texttt{dbf}_E$

Figure 5: Utility of min-composition operation.



(a) Tasks G and H are EDF schedulable



(b) $(\texttt{dbf}_s)_{34}(t) = (\texttt{dbf}_G)(t+4)$

Figure 6: Utility of left-shift operation.

DEFINITION 5 (LEFT-SHIFT). *A server S is said to implement* left-shift by τ *of a DBS S_1, denoted as $S := (\overleftarrow{}^{\tau} S_1)$, if the glue logic of S behaves as*

$$\begin{aligned} d &:= d_1 - \tau, \\ c &:= c_1, \\ r &:= \max(t, r_1 - \tau) \\ \delta_1 &:= \delta. \end{aligned} \tag{13}$$

The following result characterizes the left-shift operation. We skip the proof due to space constraints.

THEOREM 7. *Let S_1 be a DBS with demand bound function $(\texttt{dbf}_s)_1$, such that $(\texttt{dbf}_s)_1(t+\tau) \leq t, \forall t \geq 0$. Then, $S := (\overleftarrow{}^{\tau} S_1)$ defines a DBS with a demand bound function $\texttt{dbf}_s$ given as*

$$\texttt{dbf}_s(t) = (\texttt{dbf}_s)_1(t+\tau), \quad t \geq 0. \tag{14}$$

4.4 A wider class of DBS

We return to the example discussed in Section 4.1. As discussed, using existing dynamic priority servers and SP-DBS, we were unable to schedule the task set. Now, consider the min-composed server $S_{E'} := S_1 \wedge S_2$, where S_1 and S_2 are SP-DBS with parameters (1,1.5,1.5) and (1,1,2), respectively. Let task F be served by a SP-DBS S_F with parameters (1.5,6,2). As shown in Figure 5(b), the $(\texttt{dbf}_s)_{E'}$ tightly follows the demand bound function of task E, and thus with this configuration the task set is schedulable.

Consider a different example of a task set with jitter. Let G be a periodic task with jitter, with execution time 1, period 2, relative deadline 2, maximum jitter 1 for a maximum of 4 consecutive events. Task H is periodic with execution time 1, period 5 and relative deadline 2. The task set is EDF schedulable as shown in Figure 6(a). As before, there exists no configuration of existing servers or SP-DBS that serves the tasks. Further, no min-composition of SP-DBS can serve the task set. Let S_3 and S_4 be SP-DBS with parameters (1, 2, 2) and (1, 1, 6), respectively. Let $S_{34} := S_3 \wedge S_4$ and $S_G := (\overleftarrow{}^{4} S_{34})$. Then, as shown in Figure 6(b), S_G tightly serves task G. Let S_H be a SP-DBS with parameters (1, 5, 2). Clearly, S_H and S_G can serve the task set while providing task isolation.

From the examples, SP-DBS with shifted periodic $\texttt{dbf}_s$ can be min-composed and left-shifted, hierarchically, to implement a wider class of DBS. Note that the implementation overhead of the glue logic of the two composition operations is linear in the number of servers composed. Combined with the characterized implementation complexity of the SP-DBS algorithm (Section 3.4), this broadens the class of DBS with a well characterized implementation complexity. More specifically, if a task has a constant relative deadline and if its arrival curve, as defined in real-time and network calculus (9), is the shaping curve of a leaky bucket shaper [13, 17], then its demand bound function can be tightly served by a DBS implemented using the presented techniques. This range of servers can tightly serve hard real-time task sets in most practical settings.

5. CONCLUSIONS

Task isolation among real-time tasks sharing the same resource is a highly desirable feature in hard real-time systems. We illustrated with examples, that for schedulable task-sets there exist no configuration of existing dynamic priority servers that can schedule them. We proposed the Demand Bound Server (DBS) to overcome this schedulability gap. The DBS is characterized by the demand bound function $\texttt{dbf}$ which is related to the demand bound function of the task it serves. With the proposed DBS Properties I and II, we established the defining properties of such a server. We presented schedulability and worst-case response time analysis of DBS.

A major result is the optimality of the server that to our best knowledge does not hold for any other server described in the literature: For any given set of tasks with associated demand $\texttt{dbf}_t$, if there exist no demand bound servers that can serve these tasks, then there exist no other dynamic priority servers that are able to.

We then focussed on specific DBS that can be used in a realistic setting. We designed the Shifted Periodic DBS (SP-DBS) and characterized its overhead. We presented two composition operations on DBS: min and left-shift. When used hierarchically on SP-DBS, these operations greatly broaden the set of implemented demand bound functions. To the best of the our knowledge, this it the first such approach to compose multiple servers to design a single server.

Acknowledgments

This work was supported by EU FP7 projects EURETILE and PRO3D, under grant numbers 247846 and 249776.

6. REFERENCES

[1] Z. Deng and J. W.-S. Liu, "Scheduling real-time applications in an open environment," in *IEEE Real-Time Systems Symposium*, IEEE Computer Society, 1997.

[2] M. Spuri and G. C. Buttazzo, "Efficient aperiodic service under earliest deadline scheduling," in *IEEE Real-Time Systems Symposium*, IEEE Computer Society, 1994.

[3] M. Spuri and G. C. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems," *Real-Time Systems*, vol. 10, no. 2, 1996.
[4] T. M. Ghazalie and T. P. Baker, "Aperiodic servers in a deadline scheduling environment," *Real-Time Systems*, vol. 9, no. 1, 1995.
[5] L. Abeni and G. C. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *IEEE Real-Time Systems Symposium*, 1998.
[6] G. Lipari and S. K. Baruah, "Efficient scheduling of real-time multi-task applications in dynamic systems," in *IEEE Real Time Technology and Applications Symposium*, 2000.
[7] G. Lipari, J. Carpenter, and S. K. Baruah, "A framework for achieving inter-application isolation in multiprogrammed, hard real-time environments," in *IEEE Real-Time Systems Symposium* [18].
[8] T. Facchinetti, G. C. Buttazzo, M. Marinoni, and G. Guidi, "Non-preemptive interrupt scheduling for safe reuse of legacy drivers in real-time systems," in *ECRTS*, IEEE Computer Society, 2005.
[9] G. Lipari and S. K. Baruah, "Greedy reclamation of unused bandwidth in constant-bandwidth servers," in *ECRTS*, IEEE Computer Society, 2000.
[10] M. Caccamo, G. C. Buttazzo, and L. Sha, "Capacity Sharing for Overrun Control," in *IEEE Real-Time Systems Symposium* [18].
[11] L. Marzario, G. Lipari, P. Balbastre, and A. Crespo, "Iris: A new reclaiming algorithm for server-based real-time systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, IEEE Computer Society, 2004.
[12] I. Shin and I. Lee, "Compositional real-time scheduling framework," in *RTSS*, IEEE Computer Society, 2004.
[13] J.-Y. L. Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, vol. 2050 of *Lecture Notes in Computer Science*. Springer, 2001.
[14] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, vol. 4, 2000.
[15] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *IEEE Real-Time Systems Symposium*, 1990.
[16] L. Abeni, L. Palopoli, C. Scordino, and G. Lipari, "Resource reservations for general purpose applications," *IEEE Trans. Industrial Informatics*, vol. 5, no. 1, 2009.
[17] A. S. Tanenbaum, *Computer Networks*. Prentice-Hall, 1981.
[18] *Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS 2000), Orlando, Florida, USA, 27-30 November 2000*, IEEE Computer Society, 2000.

APPENDIX

PROOF OF THEOREM 1. This directly follows from (2) derived in [15] and DBS Property I. □

PROOF OF THEOREM 2. Let R' be the output trace when the given task with input trace R is served by DBS characterized by $\mathtt{dbf}_s$. We know from DBS property II that, for any $t \geq 0$, there exists $s \in [0, t]$ such that

$$\begin{aligned} & R'(t) \geq R(s) + \mathtt{dbf}_s(t-s) \\ \Rightarrow & R'(t) \geq \inf_{0 \leq s \leq t} \{R(s) + \mathtt{dbf}_s(t-s)\} \\ \Rightarrow & R' \geq R \otimes \mathtt{dbf}_s \qquad (15) \\ \Rightarrow & \mathtt{Del}(R, R') \leq \mathtt{Del}(R, R \otimes \mathtt{dbf}_s). \end{aligned}$$

Then the upper-bound on WCRT given in (7) follows. □

PROOF OF THEOREM 3. Let $\{\tau_i\}$ be a set of tasks characterized by DBF $\mathtt{dbf}_i$, respectively, such that $\sum_i \mathtt{dbf}_i(t) \leq t, \forall t \geq 0$. Let task τ_i be served by DBS_i, characterized by $\mathtt{dbf}_i$ as its demand bound function. From Theorem 1 we know that this set of DBS is EDF schedulable.

We prove that the DBS satisfies the real-time requirements of tasks by contradiction. Let some job J_j of some task τ_i arriving at time a_j miss its real-time deadline d_j, i.e. there is still some pending execution left of job J_j at time d_j. Now we can use DBS Property II, with $t = d_j$ on DBS_i, and identify an $s \leq t$, such that

$$R'(t) - R(s) \geq \mathtt{dbf}_s(t-s). \qquad (16)$$

Two cases arise: $s > a_j$ and $s \leq a_j$.
Case (a) with $s > a_j$: We know that the DBS executes jobs in FCFS order. Since at time t, job J_j that arrived at a_j has not yet completed, no job that has arrived after a_j could have been completed. Thus, we have

$$R'(t) < R(a_j + \epsilon), \text{ for any } \epsilon > 0. \qquad (17)$$

Since, $s > a_j$ and R is a non-decreasing function, we have

$$R'(t) - R(s) = R'(t) - R(a_j + (s - a_j)) < 0, \qquad (18)$$

where the second inequality comes from (17). Therefore with $t - s \geq 0$, $\mathtt{dbf}_s(t-s) < 0$, which contradicts the definition of a demand bound function. Hence, the contradiction.
Case (b) with $s \leq a_j$: We know from (16) that jobs of τ_i that have arrived and had deadline within $[s, t]$ have already executed for at least $\mathtt{dbf}_i(t-s)$. We add to this the pending execution time of job J_j at time t. Then, in $[s, t]$, the sum of execution times of jobs of τ_1 that arrived and had deadlines within $[s, t]$ *strictly exceeds* $\mathtt{dbf}_i(t-s)$. This violates the assumption that task τ_i satisfies the demand bound function $\mathtt{dbf}_i$. Hence, we have a contradiction. □

PROOF OF THEOREM 4. Let ρ be the set of 3-tuples that characterizes the sequence of requests of the SP-DBS, as defined in Section 2.2. An entry (x, y, z) in ρ implies that the server requested for z units of resource at time x to be served within a deadline y. SP-DBS requests for the shared resource when it inserts the task queue into the EDF queue in Steps 2(c) and 3(d) of the server algorithm. In either case, the absolute deadline $d \geq t + D_s$. Thus, for every tuple in (x, y, z) in ρ, y is at least $x + D_s$.

New replenishments are noted by inserting an entry into η in Step 3(a). The value of t' is always maintained to be the starting time of the last request of resource by the server. The value of $c' - c$ is maintained to be the amount of resource consumed since the last request for resource by the server. So, by inserting an entry $(t' + P_s, c' - c)$ into η, any consumed budget is earmarked for replenishment by the same amount, P_s units of time after the request. Thus, in any interval of length P_s, the amount of resource consumed can be at most the maximum capacity Q_s.

From the above two properties we have $\forall\ t, \Delta \geq 0$,

$$\begin{aligned} \sum_{\{(x,y,z) \in \rho \mid x \geq t \wedge y \leq t+\Delta\}} z \leq & \sum_{\{(x,y,z) \in \rho \mid x \geq t \wedge x \leq t+\Delta-D_s\}} z \\ \leq & \max\left\{0, \left(\left\lfloor \frac{\Delta - D_s}{P_s} \right\rfloor + 1\right) \times Q_s\right\} \\ = & \mathtt{dbf}_s(\Delta). \qquad (19) \end{aligned}$$

Hence, SP-DBS satisfies DBS Property I. □

PROOF OF THEOREM 5. Let R be the input trace of a stream of jobs served by a SP-DBS characterized by parameters (Q_s, P_s, D_s). Let R' be the output trace. Let t be any given time. Our goal is to identify an s such that

$$R'(t) \geq R(s) + \max\left\{0, \left(\left\lfloor \frac{t-s-D_s}{P_s} \right\rfloor + 1\right\} \times Q_s\right\} \quad (20)$$

If the task queue is empty at time t, then all jobs that have arrived so far have completed execution. So, we have $R'(t) = R(t)$. If we set $s = t$, (20) holds, and we are done.

Let the task queue be non-empty at time t. Let d_t be the deadline of the task queue at time t. Since, the task queue is non-empty, we have $d_t \geq t$. We consider two cases: (a) $t < d_t$, and (b) $t = d_t$.

Case (a) with $t < d_t$: The task queue deadline d is increased in Steps 2(a) and 3(b) of the algorithm. In either case, a new request for resource is made at time $d - D_s$. Since deadline at time t is d_t, there must have been a request for resource at $s' = d_t - D_s$. In $[s', t]$ the task queue would be executed for some positive amount of time. If the request at s' is because a new task arrives while the task queue is empty (Step 2(a)), then we can set $s := s'$ and show that (20) holds. Otherwise, the request was because of a replenishment (Step 3(b)), and we know that $c = 0$ at time s'. Thus, in the interval $[s' - P_s, s']$, the jobs of the task queue must have executed for the full capacity Q_s. We set $s' := s' - P_s$ and repeat the above argument. Again, if at s' a new task had arrived while the task queue was empty, we set $s := s'$ and show that (20) holds. Otherwise, we again decrease the value of s' by P_S. Let n be the number of times we reduce the value of s' by P_s, until we find s. Then, we have $s + nP_s < t < s + nP_s + D_s$. From the above argument we know that the jobs of the task queue have executed for strictly more than nQ_s time in the interval $[s, t]$. Thus, $R'(t) - R'(s) > nQ_s$. Also, at s, the task queue was empty, and hence, $R'(s) = R(s)$. From the above we have (20).

Case (b) with $t = d_t$: If the task queue is not in the EDF queue at t, we would have $t < d_t$ (Step 3(d)). Thus, at the given t, we have the task queue in the EDF queue and contending for resources. Since, $t = d_t$, it means that a request for resource made by the server completes just at its deadline. From Theorem 4, we know that all requests made by the SP-DBS are schedulable when the overall system is EDF schedulable. Thus, there must be some interval $[s'', t]$, in which the EDF schedulability constraint is tight, i.e., the total amount of resource requested by all the servers in the system in $[s'', t]$ equals $(t - s'')$. Thus, in $[s'', t]$ the total resource requested by the DBS is $\mathtt{dbf}_s(t - s'')$, i.e., $R'(t) - R'(s'') = \mathtt{dbf}_s(t - s'')$. Now we follow the same line of argument as in Case (a). Let $s' := s''$. If the request made by the server at time s' is because a new task arrives while the task queue is empty (Step 2(a)), we set $s := s'$. Otherwise, the request was because of a replenishment (Step 3(b)), and we know that $c = 0$ at time s'. Thus, in the interval $[s' - P_s, s']$, the jobs of the task queue must have executed for the full capacity Q_s. We set $s' := s' - P_s$ and repeat the above argument, until we set s. Let the n be the number of times we reduced the value of s'. Then we have $s = s'' - nP_s$, and $R'(s'') - R'(s) = nQ_s$. Since the task queue is empty at s, we have $R'(s) = R(s)$. Combining this with $R'(t) - R'(s'') = \mathtt{dbf}_s(t - s'')$, we obtain (20). □

PROOF OF THEOREM 6. To prove that S is a DBS with $\mathtt{dbf}_s$ as given in (12), we need to show that DBS properties I and II hold, given that each individual server S_i, $i \in \{1, 2, \ldots, n\}$, satisifies DBS properties I and II with demand bound functio $(\mathtt{dbf}_s)_i$.

Any expended capacity δ is accounted for by *all* the individual servers, i.e., if jobs from the task queue execute on the system, then the capacity variable of *all* servers are reduced by the amount of execution received. Also, the glue logic, sets the task queue paramters such that the requests for execution made by the server S does not exceed the requests for execution by *any* of the individual servers. Combining this with the fact that each individual server S_i, $i \in \{1, 2, \ldots, n\}$, satisifies DBS property I, we can show that S satisifies DBS property I, with $\mathtt{dbf}_s$ as given in (12).

The proof that the composed server satisfies DBS property II is more involved. To this end, we need to discuss a general implementation of a DBS: a DBS which is allowed to store all information of the past, in particular the input arrival trace. Let R be the input arrival trace of stream of jobs served by a DBS characterized by $\mathtt{dbf}_s$. Let R' be the output arrival trace of the stream of jobs after being served by the server. Let at some time t, the server algorithm be called. Then, the general DBS sets the task queue parameters as follows:

$$d := \min\big(s : (R \otimes \mathtt{dbf}_s)(s) > R'(t)\big), \quad (21)$$

$$c := (R \otimes \mathtt{dbf}_s)(d) - R'(t), \quad (22)$$

$$r := t. \quad (23)$$

DBS properties I and II follow from the above implementation. With respect to analysis of the DBS property II of the composed server, without loss of generality, we can assume that the implementation of the individual DBS is as given above. Let R be the input arrival trace of a stream of jobs and R' be the output trace of the stream when served by the composed server S. Let at time t, the server algorithms of each individual server and the glue logic of S be called. Since each individual server S_i, $i \in \{1, 2, \ldots, n\}$ is a valid DBS, paramters (d_i, c_i, r_i) by it satisfy (21-23) with $(\mathtt{dbf}_s)_i$. The glue logic of server S uses the above values and sets the task queue parameters d, c and r as in (11).

For $\mathtt{dbf}_s$ as given in (12), let $(R \otimes (\mathtt{dbf}_s)_i)(d_i) = (R \otimes \mathtt{dbf}_s)(d)$, for all $i \in \{1, 2, \ldots, n\}$. Then,

$$\begin{aligned} d &= \max_i \Big(\min\big(s : (R \otimes (\mathtt{dbf}_s)_i)(s) > R'(t)\big)\Big) \\ &= \min\big(s : (R \otimes \mathtt{dbf}_s)(s) > R'(t)\big), \quad (24) \\ c &= \min_i \big(R \otimes (\mathtt{dbf}_s)_i)(d_i) - R'(t)\big) \\ &= (R \otimes \mathtt{dbf}_s)(d) - R'(t), \quad (25) \\ r &= t. \quad (26) \end{aligned}$$

Thus, S satisfies DBS property II with $\mathtt{dbf}_s$ as in (12).

Let for some one server S_i, $(R \otimes (\mathtt{dbf}_s)_i)(d_i) < (R \otimes \mathtt{dbf}_s)(d)$. Then it follows that $d_i < d$. Since, the system is schedulable and S satisfies DBS property I, we would have a capacity of c_i consumed between $[t, d_i]$. At this point, the server algorithms would be called to set their parameters again. By repeating this argument, we would arrive at a time t, when for no server S_i, $(R \otimes (\mathtt{dbf}_s)_i)(d_i) < (R \otimes \mathtt{dbf}_s)(d)$. Then, for that t like in the previous case, we can show that S satisfies DBS property II with $\mathtt{dbf}_s$ as in (12). Thus, instead of the providing the capacity in one go, it is provided in smaller units as decided by the minimum capacity amongst all servers. However, it is provided nevertheless. □