1 Delay Bounded Aperiodic Scheduling Systems - Bag of tasks (Non-Interactive Application)

-Bharadwaj Veeravalli

Media service rendering architectures - Recent efforts in Cloud-based Content Distribution Networks (CDNs). Although Service Provider (SP) systems strive constantly and put their best efforts to minimize end-to-end delays in handling client requests, often times network support is obdurate! Hence it becomes a necessity for a SP (with high-end servers) to carefully handle these requests so that overall delay is minimized. Thus, our objective in this section is to learn a technique that a Media Server (MS) can follow to minimize the overall end-to-end delay for clients, especially when geographical distances are fairly large and routing follows multiple-hops. Note that this is clearly **not** applicable for a **real-time interactive applications**. The scenario and the solution we propose is typical of passive media applications like, stored-data rendering, graphics and/or animation presentation, yellow-books browsing, and certain proprietary applications that cannot be cached on near-client proxies, etc.

1.1 Problem context & formulation

Consider a High-End Media Server(HiEMS) supporting several clients and also is a primary server for certain proprietary stored-data rendering applications and other non-real-time data rendering applications. We consider a situation wherein a set of requests arrive to this HiEMS for media data and near-client proxies do not have these requested data. Thus for requests arriving from a proxy or a client pool connected to a proxy needs to redirect to the server concerned.

Suppose there be N requests from a client pool from a network domain at a HiEMS at a certain point in time, submitted by the respective proxy. Let us suppose that the time (estimated time often) required to process each request and the deadline (QoS!) is known. Let us define **delay** (typically lateness) in processing a request is the amount of time elapsed after its deadline. Clearly, we see that there are $N!$ sequences possible to generate different service distributions and each service distribution (sequence in which requests are served) yields different overall delays. *We attempt to choose one such sequence that minimizes the overall delay in handling all the requests.*

There are several ways of solving this problem, however, formulation needs to be more formal to understand the inherent issues in the context. Let us learn using an example. Consider the following data given to us.

Let us define a 3-tuple $< req\#, time - to - complete, deadline >$ for each request. Let us suppose we have 4 requests with the parameters as follows: $< 1, 6, 8 >, < 2, 4, 4 >, < 3, 5, 12 >, < 4, 8, 16 >$, respectively. Consider handling the requests in the sequence $1 - 2 - 3 - 4$. Then the following table gives the delays that would result for this sequence.

Job#	Completion time	Delay
1	6	0
2	$6 + 4 = 10$	$10 - 4 = 6$
3	$6 + 4 + 5 = 15$	$15 - 12 = 3$
4	$6 + 4 + 5 + 8 = 23$	$23 - 16 = 7$

So for this sequence, the end-to-end overall delay is $0 + 6 + 3 + 7 = 16$. *Our key concern is that can we identify a sequence that results in a minimum overall delay?*

1.2 Solution using Branch-and-Bound Technique

We now describe a branch-and-bound technique using 0-1 programming formulation and derive a best sequence that has a minimum overall delay. Since every possible solution to the problem must specify the order in which the requests are to be served, we define:

$x_{ij} = 1$, if req i is the j-th req to be served and $x_{ij} = 0$, otherwise.

The technique begins by partitioning all possible solutions according to the request that is last served. Note that any sequence of serving must have, $x_{14} = 1, x_{24} = 1, x_{34} = 1$, or $x_{44} = 1$. So we can think of 4 "branches" with nodes 1-4 in the Figure 1 shown below.

After creating nodes, we attempt to obtain a *lower bound* on the node delay (D) associated with that node.

In our example, if $x_{44} = 1$, we know that the request 4 is the last request to be served. Thus, request 4 would be completed at $t = 23$ units and will be delayed by 7 units. This implies any sequence having $x_{44} = 1$ must have $D \geq 7$. Thus we see $D \geq 7$ inside node 4 in the figure.

Similar analysis is carried out for other three branches. That is, $x_{34} = 1$ will have $D \geq 11$, $x_{24} = 1$ will have $D \geq 19$, and $x_{14} = 1$ will have $D \geq 15$. Note that there is no reason to exclude any node from consideration as a part of optimal request sequencing, we choose to branch on a node. We choose that node that has the lowest bound - node 4 in this example. We apply the above procedure, and follow to branch and bound the delay further. Thus, Node 9 corresponds to a sequence $1-2-3-4$, which yields a total delay of 16 time units. (*How?*) Node 9 is a feasible solution and clearly seems to be a candidate having $D = 16$. Similarly Node 8 corresponds to a sequence $2-1-3-4$ with a total delay of 12 units. Visiting every node, we see that Node 8 yields a delay of 12 units proposing a sequence $2-1-3-4$.

Thus HiEMS chooses this sequence to process the requests and retrieves the media data. Note that the sequence derived may be even used to retrieve data in a round-robin fashion for each of the requests (assuming the delays for retrieval also proportionally scale down/up).

Exercise 1: Verify Figure 1 calculations. Why the above procedure is not suitable for media streaming applications?

Exercise 2: What advantage, if any, can exist if each intermediate node until the proxy follows a computation similar to HiEMS using B-and-B technique?

Exercise 3: Consider the requests $< 1, 6, 8 >, < 2, 6, 4 >, < 3, 5, 12 >, < 4, 8, 9 >$ with their respective parameter values. Obtain a branch-and-bound solution. Is your solution optimal?