European Journal of Operational Research 270 (2018) 625–635

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Decision Support

Calculation of the performance region of an easy-to-optimize alternative for Generalized Processor Sharing

Jasper Vanlerberghe[*], Joris Walraevens, Tom Maertens, Herwig Bruneel

Department of Telecommunications and Information Processing, Ghent University, Sint-Pietersnieuwstraat 41, Ghent B-9000, Belgium

ARTICLE INFO

Article history:
Received 26 April 2017
Accepted 10 April 2018
Available online 18 April 2018

Keywords:
Queueing
Hierarchical Generalized Processor Sharing
Feasible region
Optimization

ABSTRACT

Service differentiation is a basic requirement in every modern queueing system with multiple classes of customers. In this paper, we look at Hierarchical Generalized Processor Sharing (H-GPS), which is a discrete-time hierarchically-structured implementation of the well-known idealized Generalized Processor Sharing (GPS) scheduling discipline. We prove that, for three classes, H-GPS can be configured to obtain any performance possible by other scheduling mechanisms, such as priority queueing or GPS. The hierarchical nature of a H-GPS system, however, has the major advantage that optimization is easier and more intuitive. To this end, we also present an algorithm to calculate the configuration parameters for H-GPS given a certain performance objective.

1. Introduction

In communication/computer systems, it is common to have several classes of traffic/jobs and for the administrator to desire some service differentiation between said classes. These kinds of systems can be studied from a discrete-time queueing theoretic point of view, which we do in this paper. Specifically, we consider a discrete-time queueing system with three classes of customers and a single server. These customers are a general term for what can be arriving jobs or traffic packets (depending on the system of interest) that require a certain *service* from the system. Each customer requires a certain number of slots of service from the server (possibly stochastic and class-dependent), i.e. brings a discrete number of units of work in the system. The server serves one unit of work per slot on a pre-emptive basis (it can switch to workunits of a customer of another class before finishing the work from the previous customer). Backlog of a specific class is stored in a class-specific queue which is served on a FIFO basis. In this paper, we focus on units of work left in the system (unfinished work), not individual jobs/customers. Consequently, without loss of generality and for ease of communication and brevity, we assume for the remainder of this paper each customer only requires one single slot of service. In this case, units of work and customers coincide.

For service differentiation numerous different scheduling policies exist, each with their own benefits and drawbacks. In this paper, we study a hierarchical version of the well-known Generalized Processor Sharing, called H-GPS. We first prove that H-GPS does not limit the achievable performance region of the queueing system. Secondly, we show that H-GPS is easy to optimize and present an algorithm to obtain the correct configuration of the scheduler for a desired performance setting. Before introducing H-GPS and its inner workings, we evaluate other work-conserving scheduling policies to establish a frame of reference. Work-conserving scheduling policies have the property that the server never idles when there is unfinished work in the system. Consequently, the amount of unfinished work in the system is independent from the chosen scheduling policy.

As a first policy, we consider strict priority. This is a policy whereby the classes are ordered in a hierarchical way (Atencia, 2017; Kim & Chae, 2010; Laevens & Bruneel, 1998; Öner-Közen & Minner, 2017; Rahimi & Pournaghshband, 2016; Walraevens, Fiems, & Bruneel, 2011). The server checks the queues in their hierarchical order at each time slot, serving the first non-empty queue. This way, the administrator ensures the performance of the class with top priority is not influenced by the other classes. One of the drawbacks, however, is potential starvation of lower priority classes. This happens when high load of higher level classes and their priority causes lower class customers to not get served for an extremely long time. Strict priority also has the disadvantage of not being flexible. The only way to tune performance is to reorder priorities.

[*] Corresponding author.

E-mail addresses: jasper.vanlerberghe@ugent.be (J. Vanlerberghe), Joris.Walraevens@ugent.be (J. Walraevens), Tom.Maertens@ugent.be (T. Maertens), Herwig.Bruneel@ugent.be (H. Bruneel).

https://doi.org/10.1016/j.ejor.2018.04.018
0377-2217/© 2018 Elsevier B.V. All rights reserved.

If we want to evaluate a policy, we can look at a certain performance metric, for instance the mean unfinished work of each of the classes (assuming the system is stable and such means exist). Then, the performance of the system at hand is characterised by a vector $\bar{\boldsymbol{w}} = [\bar{w}_1, \bar{w}_2, \bar{w}_3]$ of the mean unfinished work for each of the classes. For strict priority, given a certain arrival pattern of customers, it is clear that only $3! = 6$ different performance vectors can be achieved, corresponding to 6 possible priority orderings. One can intuitively see that, if we only consider work-conserving policies, the performance vector of every policy will always be included in the convex polytope with the 6 performance vectors of strict priority as vertices. This is also clarified in for instance (Bertsimas, Paschalidis, & Tsitsiklis, 1994; Dacre, Glazebrook, & Niño Mora, 1999; Gupta, Hemachandra, & Venkateswaran, 2014; Hassin, Puerto, & Fernández, 2009; Pavlin, 2017; Shanthikumar & Yao, 1992). Furthermore, Federgruen and Groenevelt (1988) have shown that every performance vector in this polytope can be achieved by a certain policy. In their paper, they construct a policy whereby in each busy cycle a different priority ordering is used. By selecting the different orderings with appropriate probabilities in the subsequent busy cycles, the policy achieves the requested performance vector in the limit (averaged over all busy cycles). In practice, this policy is not very valuable as the performance for customers of a certain class strongly depends on the busy cycle of their arrival, which is an undesired property for practitioners; this policy lacks consistency of performance in time.

Another policy option is a discrete-time version of the idealized Generalized Processor Sharing (GPS) (Anselmi & Verloop, 2011; Debicki & Mandjes, 2007; Maglaras & Van Mieghem, 2005; Parekh & Gallager, 1993; Parekh & Gallagher, 1994). This discrete-time version is strongly related to PGPS (Packet-by-packet GPS) as described in Parekh and Gallager (1993). At the beginning of each slot the server chooses which queue to serve next. Queue 1 is served with probability β_1, queue 2 with probability β_2 and queue 3 with probability $1 - \beta_1 - \beta_2$. When one or two queues are empty their probability share is redistributed proportionally to the other nonempty queues. GPS resolves the problem of starvation. Moreover, it is also flexible: choosing the weights β_1 and β_2 appropriately, any performance vector in the aforementioned polytope can be achieved. The border of the polytope (and the vertices that correspond to priority scheduling) is reached in the limit where one of the probabilities is 1 (high priority) and both others 0. In this limit case however, it is necessary to further specify the bandwith shares between the classes with 0 probability, as it is otherwise unclear which queue to serve in the event that the high priority queue is empty. The disadvantages of GPS lie in the difficult analysis of its performance and the subsequent optimization of the parameters. Obtaining these β_1 and β_2 that lead to a desired performance vector $\bar{\boldsymbol{w}}^*$ inside the polytope (so we know it is achievable with GPS), is far from straightforward. The fact that a small perturbation in one of the β_i $(i = 1, 2)$ probabilities has an influence on each of the $\bar{w}_j$ $(j = 1, 2, 3)$, makes for instance hierarchical optimization of the β_i's cumbersome.

To mitigate these analysis and optimization challenges of GPS, we investigate a hierarchical version of the aforementioned discrete-time GPS system in this paper. In the next section, we describe this H-GPS model and aim of this paper in full detail. In Section 3, we study the performance region of H-GPS for three classes. We prove that with H-GPS the whole polytope spanned by (the vertices of) priority scheduling is achievable. Subsequently in Section 4, we describe an algorithm to find the combination of input parameters to obtain a given performance vector $\bar{\boldsymbol{w}}^*$. In Section 5, we consider the implications of an extension to more than three classes. Lastly, we conclude in Section 6.

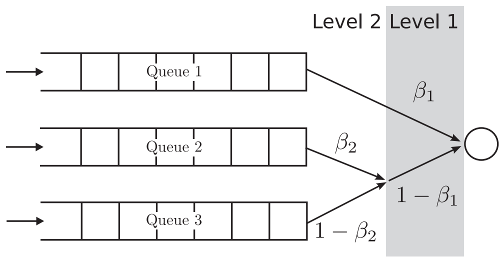

Fig. 1. Discrete-time Hierarchical Generalized Processor Sharing.

2. Model description

The hierarchical version of discrete-time GPS, that is the subject of this paper, is related to other practical implementations of the idealized H-GPS scheduling discipline, as for instance described in Floyd and Jacobson (1995), Bennett and Zhang (1997), Kalmanek, Kanakia, and Keshav (1990), and Chen and Liu (2013). In this discrete-time H-GPS version, we introduce two hierarchical levels. In each slot, assuming all queues are backlogged, the server first decides on the first hierarchical level to either serve queue 1 with probability (w.p.) β_1, or delegate service to hierarchical level 2 with probability $1 - \beta_1$. If service is delegated to level 2, queue 2 is served with probability β_2 and queue 3 is served with probability $1 - \beta_2$. If queue 1 is empty, service is always delegated to level 2. Conversely, if queues 2 and 3 are empty queue 1 is served. If service is delegated to level 2 and one of the queues on that level is empty, the other queue is served. Obviously no service happens when the system is empty. The policy is illustrated in Fig. 1. It is clear from this definition that β_2 has no influence on the decision whether or not to serve queue 1. The latter decision is taken on the first hierarchical level where β_2 plays no role. As a consequence, β_2 has no influence on $\bar{w}_1$, and thus β_1 and β_2 can be optimized hierarchically. However, it is unclear from the definition whether the performance region of H-GPS matches the polytope mentioned earlier. Therefore, we first study the performance region of H-GPS in the next section.

For completeness of the description of the dynamics of the H-GPS system, we specify the balance equations below. In these equations, $\boldsymbol{w}_k$ denotes the unfinished work at the beginning of slot k and $\boldsymbol{a}_k$ the arrivals in slot k. Analogously, $w_{j,k}$ denotes the unfinished work of class j at the beginning of slot k and $a_{j,\,k}$ the arrivals of class j in slot k.

- $\boldsymbol{w}_k = \boldsymbol{0}$
$$\boldsymbol{w}_{k+1} = \boldsymbol{a}_k \tag{1}$$
- $w_{i,k} > 0;\ w_{j,k} = 0$ with $j = \{1, 2, 3\} \setminus \{i\}$
$$w_{i,k+1} = w_{i,k} - 1 + a_{i,k}$$
$$w_{j,k+1} = a_{j,k}$$
- $w_{1,k} = 0;\ w_{2,k}, w_{3,k} > 0$
$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - [0, 1, 0] + \boldsymbol{a}_k \quad \text{w.p.} \quad \beta_2$$
$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - [0, 0, 1] + \boldsymbol{a}_k \quad \text{w.p.} \quad 1 - \beta_2$$
- $w_{2,k} = 0;\ w_{1,k}, w_{3,k} > 0$
$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - [1, 0, 0] + \boldsymbol{a}_k \quad \text{w.p.} \quad \beta_1$$
$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - [0, 0, 1] + \boldsymbol{a}_k \quad \text{w.p.} \quad 1 - \beta_1$$
- $w_{3,k} = 0;\ w_{1,k}, w_{2,k} > 0$
$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - [1, 0, 0] + \boldsymbol{a}_k \quad \text{w.p.} \quad \beta_1$$
$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - [0, 1, 0] + \boldsymbol{a}_k \quad \text{w.p.} \quad 1 - \beta_1$$