On the Design of Resource Allocation Algorithms for Low-Latency Video Analytics

Víctor Valls*, Heesung Kwon†, Tom LaPorta◇, Sebastian Stein°, Leandros Tassiulas*
Yale University*, U.S. Army Research Laboratory†, Pennsylvania State University◇, University of Southampton°

***Abstract*—In this paper, we study how to design resource allocation algorithms for data analytics services that are computationally intensive and have low-latency requirements. As a paradigm application, we consider a video surveillance service where video streams are analyzed in the cloud with deep learning algorithms (*i.e.*, object detection and image classification). We present a network model that allows data analytics to be processed in multiple stages, and propose an algorithm that provides low congestion when the arrival rate is constant over time. The algorithm also allows other types of data analytics to be carried out in the cloud in order to maximize resource utilization. The performance of the proposed algorithm is evaluated using simulation and demonstrates it is possible to obtain low-delay while maximizing the use of resources.**

I. Introduction

In this paper, we study the problem of allocating resources in the cloud to support the next generation of data analytics services. E.g., services such as video analytics that are computationally intensive and have low latency requirements. In short, the importance of providing services with low-latency is widely acknowledged by both industry and academia, and has received a lot of attention in recent years [1], [2], [3], [4]. For instance, the work in [1] emphasizes how low delay variability is crucial for providing existing and forthcoming services (*e.g.*, augmented-reality), and provides a detailed discussion of the two standard techniques used to deliver these effectively: over-provisioning and parallelization. Over-provisioning aims to have an excess of resources so that overall the system experiences very low congestion, and parallelization deals with the fact that some tasks are too large to be carried out on a single server fast enough, *i.e.*, tasks need to be divided into subtasks and processed in parallel by multiple machines. Clearly, over-provisioning reduces the delay in the system at the cost of underutilizing resources, but what is perhaps less obvious is that parallelization makes services more susceptible to experiencing significant delays. When a task is parallelized, all the subtasks must finish so that the task can be completed. And therefore, if a sub-operation gets "stuck", then the whole task is delayed. Furthermore, since the impact of parallelization gets magnified exponentially with the number of sub-operations [1], heavy over-provisioning is usually required to support low-latency services that are also computationally intensive. Next, we give an example of an application that has both characteristics.

A. Motivating application

As a paradigm application, we consider a video surveillance service where video streams are sent to the cloud to obtain a description of a scene.[1] In brief, the processing is carried out by deep-learning algorithms and consists of two stages. First, the video is fragmented into frames and an object detection algorithm decomposes each image into multiple objects. There exist many deep-learning algorithms for object detection (*e.g.*, YOLO [5] and Fast-RCNN [6]), but all of them can only detect broad classes of objects such as vehicle, person, dog, etc. Figure 1 shows an example of how a frame of a video surveillance camera in a parking lot is decomposed into four objects; a person and three cars. The second part of the processing consists of using specialized image classification algorithms to obtain a better description of the objects. For instance, we could use the image classification algorithm in [7] to find out the model of the vehicles in the parking lot. The final part of the video surveillance application is to collect the outcomes of the image classifiers and provide a description of the scene in the frame. Of course, multiple frames can also be analyzed jointly to describe the motion of the objects in the video. In system terms, the object detection and image classification tasks are carried out in a distributed manner over the servers in the cloud. There are two reasons for this primarily. First, a single server might not have access to all the deep-learning models, and second, a single machine may not be able to perform all the tasks in a short period, and so parallelization techniques are required.

Figure 1. Illustrating the detection of objects in a video frame using YOLO [5]. Once the objects have been detected, that portion of the image can be analyzed with image classification algorithms.

[1]Low-delay is crucial in video surveillance in order to be able to react to a situation promptly.

B. Challenges of resource allocation algorithms

One of the challenges of allocating resources in the cloud (or networks in general) is that it is not possible to design algorithms in an offline manner. Namely, the allocation of resources depends on the requirements of the services, and these are usually not known in advance. As a result, most of the existing resource allocation algorithms are suboptimal in the sense that they are only able to utilize a fraction of the computation and bandwidth resources available. A notable exception, however, is the case of max-weight or backpressure type algorithms [8] (with variants for data analytics [9], [10]), which can maximize the use of resources in a system without statistical knowledge of the resources required by the applications—looking at the systems' congestion is enough. However, with backpressure optimality comes at the cost of an exponential increase of the delay when the demand for resources approaches the capacity of the system. This issue is critical in many systems, but in particular in cloud computing where low-latency services compete for resources with throughput-intensive applications that are also delay-tolerant; for example, Hadoop. Specifically, throughput-intensive applications use as many resources as they can, and as a result, they increase the delay of *all* the data analytics running in the system.

C. Contributions

In this paper, we investigate how to design online resource allocation algorithms that allow low-latency services to coexist with throughput-intensive applications in the cloud. In particular, we study the case where (i) the demand of the low-latency data analytics is constant and known in advance but (ii) the other types of analytics have an unknown and time-varying demand for resources. We show that for this particular case, it is possible to design a resource allocation policy that does not create congestion for the commodities (*i.e.*, the data analytics) that have a constant demand for resources. Importantly, it is *not* possible to solve the problem in an offline manner since the demand for resources of some data analytics is not known. Our approach is motivated by the connection between queues and Lagrange multipliers (see, for example, [11, pp. 69], [12, Section 6]), and the works in [2], [3], [4] which avoid using queues to prevent congestion.

The main contributions of the paper are summarized in the following:

1) We model the problem of allocating data analytics in the cloud as a max-flow type problem with in-network processing. The model allows data analytics to be carried out in multiple computation stages and in a distributed manner.
2) We propose a resource allocation algorithm for keeping the latency low when commodities have a constant demand for network resources. From a technical point of view, the algorithm can be regarded as combining backpressure with interior-point methods in convex optimization [13].

The structure of the paper is as follows. In Section II, we review previous work on resource allocation in processing networks. Section III introduces the network model and the formulation of the resource allocation problem. Section IV presents the algorithms, and Section V evaluates their performance with an example.

II. Related Work

Most of the previous work in resource allocation in processing networks has focused on maximizing the use of resources without taking into account the delay. These works use a mix of control, optimization and queue theory, and the recurrent theme in all of them is that a subroutine or process cannot be executed if *all* the input data is not available at a processor. This is similar to video application described in Section I, where the description of a scene cannot be done unless all the image classification tasks have been completed. The authors in [10] show that by creating congestion, it is sufficient to mitigate the synchronization across subtasks in the network while maximizing the use of resources. However, the proposed algorithm does not take into account the delay and creates dummy packets which further increase the congestion in the system.

Our approach is motivated by works in [2], [3], [4], which aim to obtain low delay by not using queues. The crucial part of these approaches is to do not allocate more data analytics to a server or a communication link than what it can handle instantaneously. The recent work in [4] by Matni proposes a variation of ADMM for congestion control in data center networks that achieves zero-queueing delay. An interesting observation from the proposed algorithm is that regardless of the solution obtained with ADMM in each time slot, the amount of flow that gets into the network is truncated to never exceed the capacity of the links. As a result, a commodity can be served immediately (queues are zero) while maximizing the use of network resources.

III. Model & Problem Setup

In this section, we present the system model and formulate the resource allocation problem. In the rest of the paper, we will use the terms data analytics, commodity and flows interchangeably. We start by describing the network model with in-network processing.

A. Network model

We model a processing network as an undirected graph consisting of N nodes and L links. Let C be the set of commodities in the network and $C' \subseteq C$ the subset of commodities that have low-latency requirements. We use $x_{ij}^{(c)}$ to indicate the amount of commodity $c \in C$ allocated from node i to j, with $i, j \in N$, $i \neq j$. The allocation of a commodity in the network must respect the link capacity constraints. That is, for all $c \in C$ and $i, j \in N$ we must have

$$\sum_{c \in C} x_{ij}^{(c)} \leq \mu_{ij} \tag{1}$$

where μ_{ij} is the capacity of the link connecting node i and j. A flow allocation must also satisfy the flow conservation constraints at the nodes. We express them depending on the different types of nodes we may have in the network: source nodes (N_s), computation nodes (N_c), forwarding nodes (N_f), and destination nodes (N_d).

- **Source nodes** insert data analytics into the system that will be later processed by a computation node. The flow conservation constraints are

$$\lambda_i^{(c)} = \sum_{j \in N} x_{ij}^{(c)} \qquad \text{for all } i \in N_s, \tag{2}$$

  where $\lambda_i^{(c)}$ is the average arrival rate of a commodity $c \in C$ at node i.
- **Computation nodes** are in charge of carrying out the data analytics, *e.g.*, the object detection and the image classification. The characteristic of these nodes is that since they process data, the amount of data analytics (or flow) that arrives might not be the same as the amount that leaves the node. We capture this with the following equation

$$\sum_{j \in N} x_{ji}^{(c)} = \beta_i \sum_{j \in N} x_{ij}^{(c)} \qquad \text{for all } i \in N_c, \tag{3}$$

  where parameter $\beta_i \geq 0$ is the "gain" that a commodity experiences after it has been processed. These nodes also have the processing constraints:

$$\sum_{c \in C} \sum_{j \in N} x_{ji}^{(c)} \leq p_i \qquad \text{for all } i \in N_c, \tag{4}$$

  where $p_i \geq 0$ is the processing capacity of the node. The processing capacity constraint can also be regarded as a link capacity constraint if we replace a computation node with two forwarding nodes connected by a link, *i.e.*, the capacity of the link captures the processing capacity of the node.
- **Forwarding nodes** move data from one node to another. Their goal is to connect source nodes with computation nodes, and computation nodes with destination nodes. The flow conservation constraints for this type of node are given by

$$\sum_{j \in N} x_{ji}^{(c)} = \sum_{j \in N} x_{ij}^{(c)} \qquad \text{for all } i \in N_f. \tag{5}$$

- **Destination nodes** are, as the name indicates, where the result of the data analytics is sent after the processing. We assume that when the data analytics arrives at this node they leave the system.

B. Computation graph for the commodities

We consider the case where commodities need to be processed in a particular order. In the case of the video-surveillance application, this corresponds to extracting first the objects from the frames in the video stream, and then analyzing each of them with a specialized image classification algorithm. To capture the order in which a commodity needs to be processed, we define a set of computation graphs or virtual network embeddings (VNEs) $E(c)$ for each commodity $c \in C$. Specifically, an embedding in $E(c)$ is a collection of nodes from N and links from L that indicate the path/s that a commodity can follow from the source node to the computation nodes, and from these to the destination node. We make the usual assumption that VNEs are directed and acyclic graphs. Figure 2 shows an example of a network with one commodity and three VNEs.

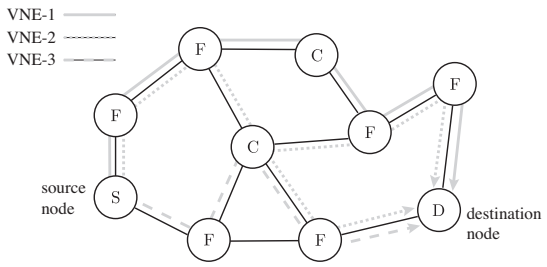

Figure 2. Illustrating an example of a processing network with one commodity and three VNEs (indicated with gray lines). The black lines indicate the connection between nodes. Observe that the three embeddings start at the source node (S), go through forwarding (F) and computation nodes (C), and end at the destination node (D). All the VNEs are directed and acyclic graphs.

We use $x_{ij}^{(e,c)}$ to denote the amount of commodity assigned to a commodity $c \in C$ in an embedding $E(c)$. The following equation must be satisfied for all embeddings

$$\sum_{e \in E(c)} x_{ij}^{(e,c)} = x_{ij}^{(c)}. \tag{6}$$

To capture that it is not possible to use some links in the network (*i.e.*, links that do not belong to an embedding), we add the constraint

$$x_{ij}^{(e,c)} \leq \mu_{ij}^{(e)}, \tag{7}$$

where $\mu_{ij}^{(e)}$ is the maximum amount of commodity $c \in C$ that can be transmitted over the link connecting nodes i and j and embedding $e \in E(c)$. Note that if $\mu_{ij}^{(e)} = 0$ then the link cannot be used by the embedding. The design of VNEs is strongly connected to the placement of functions in networks [14] and out of the scope of this paper.

C. Problem formulation

In the static or fluid version of the problem, the resource allocation problem is to find an allocation $x_{ij}^{(e,c)}$ with $c \in C$, $e \in E(c)$, $i, j \in N$ that satisfies the constraints given in the previous section. The problem can be formulated as a linear program since the equality and inequality constraints are linear, and so solved with standard linear programming techniques such as the simplex method [15]. The dynamic version of the problem is more challenging since the arrival rate (*i.e.*, parameter $\lambda^{(c)}$) of the commodities may not be known in advance. Hence, the problem cannot be solved in an offline mode, and decisions need to be taken as the state of the system evolves. In the next section, we present two algorithms that solve the dynamic version of the problem using a similar approach than in [11].

Algorithm 1 Backpressure-type algorithm

Initialize $Q_{ij}^{(e,c)}[1] = 0$ for all $i, j \in N$, $c \in C$ and $e \in E(c)$.
for $k = 1, 2, \dots$ **do**

(1) Select transmission rates: for each node $i \in N$ and link (i, j), select the $x_{ij}^{(e,c)}[k]$ that maximizes

$$\sum_{c \in C} \sum_{e \in E(c)} \left(Q_i^{(e,c)}[k] - Q_j^{(e,c)}[k]\right) x_{ij}^{(e,c)}[k] \qquad (8)$$

subject to (1), (4), (6) and (7).

(2) Update congestion variables: for each link (i, j), commodity $c \in C$, and embedding $e \in E(c)$, update the congestion variable

$$Q_i^{(e,c)}[k+1] = \Big(Q_i^{(e,c)}[k] + \Delta_{ij}^{(e,c)}[k]\Big)^+$$

(3) Share congestion variables: each of the nodes must send the congestion variables to its neighbors.

end for

IV. Resource allocation algorithms

We present two algorithms: a backpressure-type algorithm that allocates data analytics based only on the congestion in the system, and a backpressure + interior-point method algorithm that can exploit the fact that some commodities have constant demand for resources. The time in the system is divided in time slots $k \in \{1, 2, \dots\}$, and we parameterise the variables in the system with index $[k]$. Hence, $x_{ij}^{(c)}[k]$ indicates the allocation of commodity c between nodes i and j in time slot k. For both algorithms, we define a congestion variable $Q_i^{(e,c)}[k+1]$ for every node $i \in N$, commodity $c \in C$ and embedding $e \in E(c)$. These congestion variables can be regarded as queues or Lagrange multipliers, and are updated in each iteration as follows:

$$Q_i^{(e,c)}[k+1] = \Big(Q_i^{(e,c)}[k] + \Delta_{ij}^{(e,c)}[k]\Big)^+$$

where

$$\Delta_{ij}^{(e,c)}[k] = \sum_{j \in N} x_{ij}^{(e,c)}[k] - \sum_{j \in N} x_{ji}^{(e,c)}[k],$$

for the computation[2] and forwarding nodes, and

$$\Delta_{ij}^{(e,c)}[k] = \lambda_{ij}^{(c)}[k] - \sum_{j \in N} x_{ji}^{(e,c)}[k]$$

for the source nodes. It is important to emphasize that in this paper $Q_i^{(e,c)}$ may not be discrete-valued, and so it cannot be regarded as queues containing quantities such as packets. Also, note that we have constrained $Q_i^{(e,c)}[k+1]$ to be nonnegative to avoid having "negative" congestion.

A. Backpressure type algorithm

This algorithm is a variation of the backpressure-type algorithm [11] which can maximize the use of resources in a network whenever there exists an algorithm that would do so.[3] The procedure is described in detail in Algorithm 1 and consists of three steps. In every time slot, each node observes the amount of commodity in the congestion variables and selects to allocate the commodities that maximize the weighted sum in (8). Note that the allocation depends on the capacity of the links and the available VNEs, and that these can be carried out in a distributed manner.[4] The second step of the algorithm is to update the congestion variables, which is the result of the flow allocation or transmission in a time slot. The third and final step is to share the congestion variables amongst the neighboring nodes.

B. Backpressure + interior-point type algorithm

This algorithm extends the previous one and separates the allocation of the commodities in C' and $C \setminus C'$. The procedure is described in detail in Algorithm 2. In short, the key part of the algorithm is to find an allocation for the commodities in C' that satisfies the flow conservation constraints described in Section III, and *at the same time* find a max-weight type allocation for the commodities in $C \setminus C'$. To ensure that the algorithm provides no congestion to the commodities in C', we need to make the following assumption.

Assumption 1. *The commodities in C' have a constant demand for bandwidth and computation resources,* i.e.*, $\lambda_i^{(c)}[k] = \lambda_i^{(c)}[k+1]$ for all time slots $k \in \{1, 2, \dots\}$, $c \in C'$.*

One of the consequences of the algorithm is that since the flow conservation constraints are always satisfied, we have that

$$\Delta_{ij}^{(e,c)} = 0 \qquad \forall c \in C', e \in E(c)$$

and so the congestion variables of these commodities are equal to zero for all $i \in N$. An important difference from the previous algorithm is the allocation of resources now must be carried out by a centralized node. This is because, for all the commodities in C', the flow conservation constraints must be satisfied from the source node to the destination node.

Figure 3 illustrates, schematically, the intuition behind Algorithm 2. The two circles indicate the set of allocations for which the flow conservation constraints will be satisfied for the commodities in C' and $C \setminus C'$, individually. The intersection of the two sets contains the network capacity region for all the commodities in C. The crosses in the figure indicate possible allocations that Algorithm 1 could select. Observe that these do not need to satisfy the flow conservation constraints for either of the commodities. This is in contrast to the behavior of Algorithm 2 where the allocation of commodities (dots) are always in the set where the flow conservation constraints are satisfied for the commodities in C'. Importantly, recall that in Algorithm 2 we have assumed that the arrival process of the commodities in C' is known and constant, which is not the case for the commodities in set $C \setminus C'$. Also, note

[2]To streamline exposition, we assume that $\beta_i = 1$ for all $i \in N_c$.

[3]The algorithm needs that the arrival process of each commodity is i.i.d. and that the maximum amount of flow that can arrive at the source node is bounded.

[4]For a fully distributed implementation, the processing constraints need to be modeled with two forwarding nodes connected by a link.

Algorithm 2 Backpressure + Interior-point-type algorithm

Initialize $Q_{ij}^{(e,c)}[0] = 0$ for all $i, j \in N$, $c \in C$ and $e \in E(c)$.
for $k = 1, 2, \ldots$ **do**

(1) Select transmission rates: for each node $i \in N$ and link (i, j), select the $x_{ij}^{(e,c)}[k]$ for $c \in C'$ that satisfies constraints $(1)-(7)$ and at the same time select the $x_{ij}^{(e,c)}$ with $c \in C \setminus C'$ that maximize

$$\sum_{c \in C \setminus C'} \sum_{e \in E(c)} \left(Q_i^{(e,c)}[k] - Q_j^{(e,c)}[k]\right) x_{ij}^{(e,c)}[k]$$

subject to (1), (4), (6) and (7).

(2) Update congestion variables: for each link (i, j), commodity $c \in C \setminus C'$, and embedding $e \in E(c)$, update the congestion variable

$$Q_i^{(e,c)}[k+1] = \left(Q_i^{(e,c)}[k] + \Delta_{ij}^{(e,c)}[k]\right)^+$$

(3) Collect congestion variables: each of the nodes must send the congestion variables to a *central* node.

end for

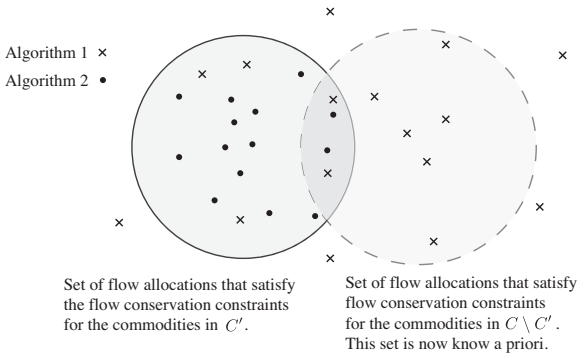

Figure 3. Illustrating the intuition behind Algorithm 2.

that the problem cannot be solved in an offline manner since the set that satisfies the flow conservation constraints for the commodities $C \setminus C'$ is not known a priori. We show this and evaluate the performance of the algorithms in the next section.

V. Numerical Example

We illustrate the performance of the algorithms in the previous section with a simple example. Consider the network in Figure 4, which consists of 6 nodes and 7 undirected links. There are two data analytics services in the system: a video surveillance service (VS) and database service (DB). The VS service consists of analyzing a video stream generated by a network of cameras with at computer vision algorithm, and the DB service answers data analytics queries from users (see Figure 4). Our task is to allocate resources in the network to allow the video stream to reach the computer vision algorithm with "zero" delay, but also to maximize the number of queries that the database can serve. To streamline exposition, we normalize the capacity of the links to 1 and consider that the video stream has a *constant* rate of $1/2$.

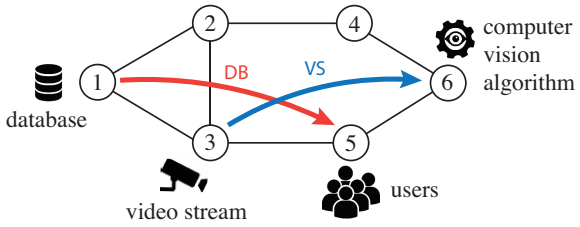

Figure 4. Illustrating the setup of the example in Section V. The video surveillance (VS) service sends a video stream from node 3 to 6, and the database service (DB) service from node 1 to 5.

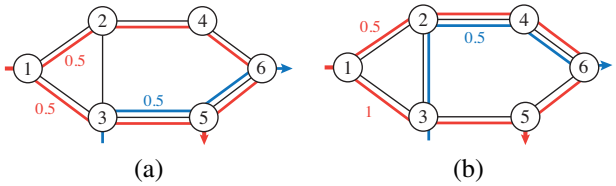

Figure 5. Illustrating how two static allocations of the VS service (blue) affect differently the achievable throughput of the DB service (red).

Next, we show (i) that an offline algorithm or static allocation of the VS service can be throughput suboptimal; (ii) that the backpressure algorithm maximizes resource utilization but does not keep the latency low when the network load is high; and (iii) that the proposed hybrid algorithm achieves both, low delay for the VS service and maximum resource utilization.

A static allocation provides zero delay to the VS service but can be throughput suboptimal: Since the VS service generates a video stream with a constant rate of $1/2$, it is possible to reserve bandwidth for the service (*e.g.*, with RSVP). For instance, from node 3 to node 6 through node 5, *i.e.*, $(3 \to 5 \to 6)$. Reserving bandwidth allows the VS service to do not suffer congestion,[5] but it may lead to suboptimal configurations in terms of total resource utilization. We show this with the example in Figure 5. Observe from Figure 5a that with the static allocation of the VS service $(3 \to 5 \to 6)$, the maximum query rate of the DB service is 1 since the VS service has reserved $1/2$ of the bandwidth in links $(3, 5)$ and $(5, 6)$. In contrast, observe that if we allocate the VS service through the path $(3 \to 2 \to 4 \to 6)$, then the DB service can achieve a rate of 1.5 (see Figure 5b). Hence, the path $(3 \to 2 \to 4 \to 6)$ is preferable in terms of system utilization. However, to find the best static allocation for the VS service one needs to know the demand for resources of the other data analytics in the system, which is something that is usually not known.

A backpressure approach is throughput optimal but does not provide zero delay to the VS service: We show using simulation that the backpressure algorithm maximizes the utilization of resources, but that the average delay or congestion of both data analytics increases exponentially when the network approaches its capacity. We consider that the VS service has a constant rate of $1/2$ and that users generate

[5] As long as the rate does not exceed the reserved bandwidth.

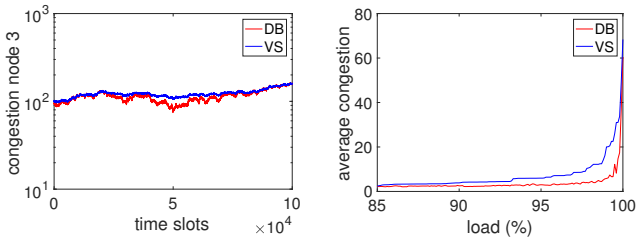

Figure 6. Illustrating the congestion in the system with the backpressure algorithm (Algorithm 1).

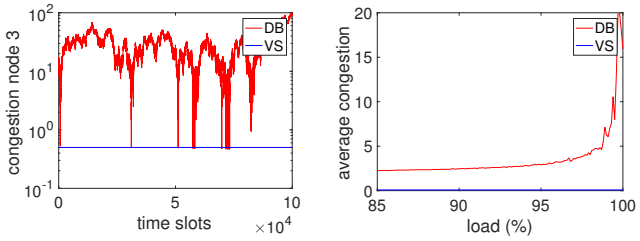

Figure 7. Illustrating the congestion in the system with the backpressure + interior-point method algorithm (Algorithm 2). The congestion of the VS service is equal to $1/2$ in both figures.

queries to the database following a poisson process. Figure 6 shows the size of the congestion variables at node 3 when the load in the system is 99% (left), and the average congestion in the system depending on the system's load (right).[6] Observe from the figure that both data analytics experience congestion and that the average congestion increases rapidly when the load is high. This means, for example, that if the DB service saturates the network then the VS service will start experiencing large delays. In this scenario, to obtain low *average* delay, it would be necessary to avoid loads over 95%.

Combining dynamic "static" allocations (interior-point method) and backpressure: zero delay for the VS service and maximum throughput. With the same setup, we run the backpressure + interior-point method algorithm (Algorithm 2) and illustrate the results in Figure 7. Observe that now the VS service experiences no congestion. Or to be more precise, the congestion variables of the VS service have size $1/2$, which corresponds to the data analytics arriving and leaving immediately. The key of Algorithm 2 is that it allocates resources to the VS service without violating the flow conservation constraints. And as a result, the congestion variables cannot grow more than the rate. It is important to emphasize that Algorithm 2 works because the arrival rate of the VS service is constant. This assumption is strong in general but reasonable for video analytics.

VI. Conclusions

We have studied how to design resource allocation algorithms for data analytics services that are computationally intensive and have low-latency requirements. We have presented a network model that allows data analytics to be processed in multiple stages, and proposed an algorithm that provides low congestion when the arrival rate is constant over time. The algorithm also allows other types of data analytics to be carried out in the cloud in order to maximize resource utilization. The performance of the algorithm is evaluated using simulation and compared to a backpressure type algorithm.

[6]The queries to the database increase to reach the full capacity of the system. The load of the VS service remains constant.

References

[1] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[2] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. M. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Queues don't matter when you can jump them!" in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 1–14.

[3] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized "zero-queue" datacenter network," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 307–318.

[4] N. Matni, "Optimal zero-queue congestion control using admm," in *American Control Conference (to appear)*, 2017.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 779–788.

[6] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Available: http://arxiv.org/abs/1504.08083

[7] J. W. Hsieh, L. C. Chen, and D. Y. Chen, "Symmetrical surf and its applications to vehicle detection and vehicle make and model recognition," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 1, pp. 6–20, Feb 2014.

[8] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, Dec 1992.

[9] M. J. Neely and L. Huang, "Dynamic product assembly and inventory control for maximum profit," in *49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 2805–2812.

[10] A. Destounis, G. S. Paschos, and I. Koutsopoulos, "Streaming big data meets backpressure in distributed network computation," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 2016, pp. 1–9.

[11] L. Georgiadis, M. J. Neely, L. Tassiulas *et al.*, "Resource allocation and cross-layer control in wireless networks," *Foundations and Trends® in Networking*, vol. 1, no. 1, pp. 1–144, 2006.

[12] R. Srikant and L. Ying, *Communication networks: an optimization, control, and stochastic networks perspective*. Cambridge University Press, 2013.

[13] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[14] D. Arora, M. Bienkowski, A. Feldmann, G. Schaffrath, and S. Schmid, "Online strategies for intra and inter provider service migration in virtual networks," in *Proceedings of the 5th International Conference on Principles, Systems and Applications of IP Telecommunications*, ser. IPTcomm '11. New York, NY, USA: ACM, 2011, pp. 10:1–10:11.

[15] G. B. Dantzig, *Linear Programming and Extensions*. Princeton University Press, 1963.