Distributed resource management using iterative gradient update synthesis

Mårtensson, Karl; Vladimerou, Vladimeros

2011

Link to publication

Citation for published version (APA):

Total number of authors: 2

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain.
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
Distributed resource management using iterative gradient update synthesis

Karl Mårtensson and Vladimeros Vladimerou

Abstract—We consider load balancing on a network. Servers of limited bandwidth move a single commodity through a network of buffers (or queues) while external random processes generate and consume this commodity. Our contribution is a distributed algorithm for regulating the backlogs of these queues to a given reference while balancing the mean flow in the network.

We formulate this as a fluid buffer regulation problem and use distributed gradient descent to update the feedback gains for an LQG controller. Our proposed distributed algorithm both implicitly and explicitly estimates the statistics of the external process flows using only local information on fixed time intervals and updates the feedback matrix for the regulator accordingly.

We demonstrate our method on a simulation of an industrial floor where autonomous vehicles remove palettes from production line buffers.

I. INTRODUCTION

Flow optimization and load balancing problems presented on computer queuing networks, can be examined from a discrete or continuous point-of-view, using algorithms on graph structures. Given a graph, the interconnection of sources and destinations, linear programming [6], [5] and fast approximate solutions can be applied for solving single- or multi-commodity flow maximization problems [7].

Also related to flow maximization is the problem of load balancing, where many servers partake in servicing incoming tasks and the goal is to maintain a load balance so as to not overload some of the queues while allowing others to idle. In [1], the authors used Discrete Event Systems models on a metric state space and Lyapunov theory for the analysis of load balancing systems. In [9] the authors also deal with load sharing in terms of buffered systems, in the same DES framework, with optimal solutions to the allocation of bandwidth and buffer size.

In Fig. 1 we see the configuration where a network of three servers is connected with 7 queues. In the work by [13] the models allow the servers to route jobs from each queue to another.

From some queues, the servers can consume the jobs, getting them outside of the system. Traffic with specified destination gets generated at the queues with some random rates (usually with bounded burst rates), and the goal is to balance/stabilize the queues and route traffic to its destination where it gets consumed, or converted to another type of traffic.

Other work considers discretely enabled links in order to avoid interference [3]. Distributed weighted graph matching algorithms are the main method used in that and similar work. Longest queue first (LQF) schemes also fall in the category of discrete service activation. Local pooling [4] properties have been long used for providing guarantees for stability of such schemes. Stability proofs follow from Lyapunov theory as the longest queue becomes then the Lyapunov function for the system.

Quite similarly, the backpressure algorithm [13], in various forms, uses an intuitive method of feedback that gives stability of the queues. For example in Fig. 2 we see that server $B_1$ is allocated to one of queues 1, 2, 3. The filled areas in the figure represent quantitatively the backlogs of the queues. Jobs are routed according to the differential “pressure” induced by the size of the queue backlogs at the servers’ sources and destinations.

Fig. 2. Backpressure algorithm intuition: Server $i$ serves queues $B_i = \{1, 2, 3\}$. Traffic from queue 2 can be routed by server $i$ to any queue in the set $R_2 = \{5, 6, 7\}$. In this example, with the queue backlog as the filled area in each queue, server $i$ will choose queue 2 to service and will direct its traffic to buffer 7.

We present a combined flow optimization and load bal-
ancing algorithm. Our work considers the allocation of server resources in routing identical items from their sources to possible destinations through a network of buffers (or queues) so as to balance the queue backlogs and avoid overflows. Our model is very similar to the classic one in [13] with one of the main differences being that it is formulated on a continuous state space to begin with. Unlike the above references which use, in the heart of their algorithms, differential queue comparison, we use a distributed LQG regulator, which allows for very fast localized computation. At the same time, in longer intervals, we update the statistics and improve the performance of our feedback rule using a distributed gradient descent algorithm introduced in [10], [11] and a linear program. Our method has fast computation times so it can be applied to both vehicular traffic as well as digital computer networks.

The rest of this article is structured in sections as follows:

(II) after this introduction we give a preliminary overview of our distributed iterative learning control (ILC) scheme
(III) we formulate the problem on a networked buffer model and give our complete algorithm
(IV) we demonstrate the effectiveness of our algorithm in an example application: an industrial multi-vehicle system for handling product palettes
(V) we conclude with some remarks and future plans

II. DISTRIBUTED ITERATIVE LEARNING CONTROL

In this section we will summarize the theory behind the method for distributed ILC, found in [10], [11]. The systems considered have an associated graph, such that each vertex is a partition of the system states in such a way that each vertex represents a subsystem of the complete system. An edge between two vertices indicates that two subsystems directly affect each other. The edges also specify with which subsystems a certain vertex can communicate. Consider a discrete time system in state space form

\[ x[n+1] = Ax[n] + Bu[n] + w[n] \]  

where \( w \) is independent, zero-mean, Gaussian noise with variance \( \sigma_w^2 \). Let the associated graph be \( G = (V,E) \) where \( V \) is the set of vertices and \( E \) the set of edges (i.e. ordered pairs of vertices). Then there will be a partition of the states over the vertices, \( x = [x_1^T \ldots x_N^T]^T \) (we write \( x_i \) for the states associated to vertex \( v_i \)). According to the edges of the graph, we restrict the blocks of the dynamics matrix \( A \) by

\[ A_{ki} \neq 0 \implies (v_i,v_k) \in E \text{ or } (v_k,v_i) \in E \]

We also assume that each subsystem has a distinct set of control signals, implying that the \( B \) matrix will be block diagonal.

The control scheme will be state feedback, i.e. \( u[n] = -Lx[n] \). In a similar manner as with the dynamics matrix, we impose a restriction on the allowed feedback matrices \( L \)

\[ L_{ki} \neq 0 \implies (v_i,v_k) \in E \text{ or } (v_k,v_i) \in E \]

The structure of the feedback matrix implies that for a subsystem to calculate its control signal, it is allowed to use measurements from other subsystems only if those are its neighbors, i.e. there is a edge between them in the graph.

We introduce the performance

\[ J(L) = E(\|x_0^2\| + |u|^2_R) \]

where \( x \) satisfies (1) in stationarity with \( u = -Lx \). The weight matrix \( Q \) is assumed to be block-diagonal, positive semi-definite and \( R \) is assumed to be block-diagonal positive definite. Note that the performance \( J \) is only defined for stabilizing feedback matrices \( L \).

The objective of the distributed ILC scheme is to use measurements of \( x \) and \( u \) from neighboring agents to update the feedback matrix to improve the performance \( J \). This will be done by determining a descent direction \( G \) of \( J \) with respect to \( L \) and change \( L \) according to

\[ L^{\text{new}} = L + \gamma G \]

where \( \gamma \) is sufficiently small. All calculations are to be performed in a localized manner, i.e. to update the feedback in agent \( x_i \), only local measurements and local model information may be used.

In order to find a descent direction of \( J(L) \), we determine the gradient of \( J(L) \).

**Proposition 1**: Given a stabilizing state feedback \( u[n] = -Lx[n] \) to the system (1), the gradient of \( J \) with respect to \( L \) is

\[ \nabla_L J = 2RL - B^T P(A - BL)X \]  

where \( X \) and \( P \) satisfy the Lyapunov equations

\[ X = (A - BL)X(A - BL)^T + \sigma_w^2 \]  

\[ P = (A - BL)^T P(A - BL) + Q + L^T RL \]  

**Proof.** We use the fact that \( J = \text{tr}(P \sigma_w^2) \). Denote

\[ A_L = A - BL \]

\[ M = dL^T (RL - B^T PAL) \]

By differentiating (4) w.r.t. \( L \) we see that \( dP \) satisfies the Lyapunov equation

\[ dP = A_L^T dPAL + M + M^T \]

Hence,

\[ dP = \sum_{k=0}^\infty (A_L^T)^k (M + M^T)A_L^k \]

and

\[ dJ = \text{tr}(dP \cdot \sigma_w^2) = 2\text{tr} \left( M \sum_{k=0}^\infty A_L^k \sigma_w^2 (A_L^T)^k \right) = 2\text{tr} (dL^T (RL - B^T PAL)X) \]

We conclude the proof by using the fact that \( dZ = \text{tr}(dX^T Y) \Rightarrow \nabla X Z = Y \) for matrices \( X^T, Y \in \mathbb{R}^{n \times p} \). \( \square \)
By introducing adjoint (or dual) state variables, (2) can be rewritten so that the gradient in can be computed in a distributed way.

**Proposition 2:** Under the assumptions of Proposition 1, let the stationary process \( \lambda \) defined by the backwards iteration

\[
\lambda(t-1) = (A - BL)^T \lambda[n] - (Q + L^T RL)x[n]
\]

where \( x[n] \) is the state of the original system (\( \lambda \) is called the adjoint or dual state). Then

\[
\nabla_L J = 2(\mathbf{RL} \Sigma x^T + B^T \mathbf{E} \lambda x^T)
\]

**Proof.** Denote \( Q_L = Q + L^T RL \). The dynamics equation of \( \lambda \) gives

\[
\lambda[n] = -\sum_{j=n+1}^{\infty} (A_L^j)^{-j-1}Q_Lx[j] =
\]

\[
= -\sum_{j=0}^{\infty} (A_L^j)^{-j-1}Q_Lx[j] +
\]

\[
+ \Psi \{ (w[n], w[n+1], \ldots) \}
\]

where the operator \( \Psi \) is the appropriate linear operator of how the noise \( w[k] \) affects \( \lambda[n] \) for \( k \geq n \). Hence

\[
E\lambda[n]x[n]^T = -E \left( \sum_{j=0}^{\infty} (A_L^j)^{-j-1}Q_Lx[j] \right)^T =
\]

\[
-PA_L X
\]

since \( w[k] \) and \( x[n] \) are independent for \( k \geq n \). \( \square \)

By running the system for a number of steps and simulating the adjoint equations backwards in time, each subsystem can collect measurements of local states \( x \) and local adjoint states \( \lambda \). These can then be used to determine estimates of the variances and cross-covariances in (6) and hence determine an estimate of a part of the gradient of \( J \) used to update the part of the feedback matrix relevant to that subsystem. In subsystem \( v_1 \) the parts of the relevant feedback update \( L_i = [V_1 \lambda]^T \) where \( (v_1,v_k) \in \mathcal{E} \). Further analysis on how this procedure actually is distributed, i.e. all computations can be performed while using the allowed communication links, can be found in [10].

### III. MODEL AND DESIGN DESCRIPTION

Our system is described by a directed bigraph \( \mathcal{G} = (\mathcal{V}, \mathcal{E}) \) with vertex set \( \mathcal{V} = S \cup Q \), where \( S \) is a set of servers or routers, and \( Q \) is a set of queues or buffers. We use a continuum of fluid material to represent items routed by the servers from queue to queue. At each queue \( i \in Q \) based on a stationary discrete-time random process \( w_i[n] \in \mathbb{R} \), a continuum of fluid is produced (\( E(w_i) \geq 0 \)), or consumed (\( E(w_i) < 0 \)) every time instant \( n \). Each server \( j \in S \) has a bandwidth capacity \( C_j \) which allows it to redirect fluid among different queues. More specifically, at each time instant, it can move some fluid from any queue in its incoming set \( Q_{\text{in}}^i = \{ i|(i,j) \in \mathcal{E} \} \subset Q \) to any queue in its outgoing set \( Q_{\text{out}}^i = \{ k|(j,k) \in \mathcal{E} \} \subset Q \) for up to a total of \( C_j \) units of fluid. Similarly, each queue is characterized by an incoming set \( S_{\text{in}}^j = \{ j|(j,i) \in \mathcal{E} \} \subset S \) and an outgoing set \( S_{\text{out}}^j = \{ k|(i,k) \in \mathcal{E} \} \subset S \) of servers, which can reroute its fluid backlog. For shorthand we define \( Q_j = Q_{\text{out}}^j \cup Q_{\text{in}}^j \) and \( S_i = S_{\text{out}}^i \cup S_{\text{in}}^i \). We also define \( S_k = \{ i|\exists j \cdot j \in S_k \land j \in S_i \} \), the set of peers of queue \( k \) (other queues that share servers).

When \( E(w_i) \geq 0 \) \( (E(w_i) < 0) \) then we say that queue \( i \) is an input (output) node. This is shown in Fig. 3.

![Fig. 3](image_url)

**The system equations are thus:**

\[
q_i[n+1] = q_i[n] + w_i[n] - \sum_{j \in Q_{\text{out}}^i} \sum_{k \in Q_{\text{in}}^j} u_{jk}[n] + \sum_{j \in S_{\text{in}}^i} \sum_{k \in Q_{\text{out}}^j} u_{ji}[n] \quad \text{(7)}
\]

where \( u_{jk}[n] \) is the amount of fluid routed at time \( n \) by server \( j \) from queue \( i \) to queue \( k \), and \( w_i[n] \) is Gaussian \( \sim N(\mu_i, \sigma_i^2) \) for every \( i,n \), indicating an uncertain production/consumption rate. One can consider infinite capacity queues, or queues with an upper backlog limit \( 0 \leq q_i[n] \leq Q_i \), \( \forall i,n \). In addition, one can either assume a closed system where fluid only enters and leaves the system via the random processes \( w_i \), or an open system where some queue is allowed to have an infinite backlog (positive or negative). Using the latter one can model situations where a external resource can consume or produce fluid in the network at rates much higher than the processes in the queues.

Given that we have finite routing capacity for each server, i.e. that

\[
\sum_{(i,k) \in Q_{\text{out}}^i \times Q_{\text{in}}^j} u_{jk} \leq C_j, \text{ for some } C_j > 0 \text{ for each } j \in S,
\]

the goal is to design the process \( u_{jk}[n] \) so that the variance of \( q - q_{\text{nom}} \), for some given reference \( q_{\text{nom}} \) is minimal. In other words, we would like to control the backlogs of the queues to be as close to some fixed value as possible.

### A. Nominal Operating Points

The first step in designing a controller for our system is to remove the nonzero means from the random production/consumption processes \( w_i[n] \), so that we bring the problem into an LQG formulation. One has to assume that the nominal consumption/production rates are sustainable given the capacities of the servers. This can be verified via max-flow or feasibility analysis [6], [5] using a simple linear program.
After part of the server capacity is used to balance out the nonzero-mean noise, the rest can be allocated to regulate fluctuations in the system fluid. Without feedback for variance regulation, the queues will have a zero mean but unfortunately, the open-loop variance of their backlogs grows to infinity linearly with \( n, E[(q[n] - \mu)q[n]] - \sigma^2 n \).

An intuitive way to distribute server capacity towards variance regulation is to partition a static bandwidth \( u_{jik} \) of each server \( j \) allocated to each link \( i \rightarrow j \rightarrow k \) into: \( u_{jik} = u_{jik}^{\text{nom}} + u_{jik}^{\text{slack}} \) where \( u_{jik}^{\text{nom}} \) is used to remove the nominal mean of the exogenous inputs \( w_i \) and the slack \( u_{jik}^{\text{slack}} \) is used for variance regulation. This we can do using the following linear program:

\[
\max \min_{u_{jik}^{\text{nom}}, \gamma} \left( \sum_{j \in S} \sum_{k \in Q_k} u_{jik}^{\text{slack}} + \sum_{j \in S} \sum_{i \in Q_i} u_{jii}^{\text{slack}} - \sigma_i \right) \quad (8)
\]

\[
u_{jik} = u_{jik}^{\text{nom}} + u_{jik}^{\text{slack}} \quad \forall j \in S, \ i, k \in Q \quad (9)
\]

\[
\mu_i - \sum_{j \in S} \sum_{k \in Q_k} u_{jik}^{\text{nom}} + \sum_{j \in S} \sum_{i \in Q_i} u_{jii}^{\text{nom}} = 0 \quad \forall i \in Q \quad (10)
\]

\[
\sum_{i \in Q_i} u_{jii}^{\text{nom}} \leq C_j \quad \forall j \in S \quad (11)
\]

\[
u_{jik}^{\text{nom}}, \nu_{jik}^{\text{slack}}, u_{jik}^{\text{nom}} \geq 0 \quad \text{if } (i, j, (j, k) \in E, \text{ zero else} (12)
\]

The cost function to be maximized in (8) is the minimum difference between the sum of the incoming and outgoing slack component of the server bandwidth over all queues. The constraint (10) allocates the nominal components to balance the mean flow in the system and together \( q_{\text{nom}} \) gives the nominal operating point for the system. This allows us to eliminate the effect of the non-zero means \( \mu_i = E[w_i] \) of the external flows in/out of the system, while, at the same time, allocating enough bandwidth to handle fluctuations where the noise has large variance \( \sigma_i \). Constraint (11) gives the capacity bound for each server, and (12) gives the graph structure.

This program is used to calculate a “DC” component of the server bandwidth as shown in Fig. 4. It may be run periodically as the statistics of the production/consumption processes change. Sparsity of the system graph implies that one can use distributed constraint consensus algorithms such as the one in [12].

The quantization block in Fig. 4 is used to indicate a countable nature of the commodity - discrete items instead of continuous fluid. The saturation block is used to enforce the capacity limitation of the servers. In practice, saturation is avoided by the maximization of the minimum slack component of our servers together with the appropriate choice of \( Q \) and \( R \) matrices in the following LQG formulation. When saturation (seldomly) happens, we distribute the entire capacity of server \( j \) while maintaining the ratios belonging to each component \( u_{jk} \); for example if \( u_1 = u_{112} + u_{113} + u_{114} \) where \( u_{112} = 1, u_{113} = 3, u_{114} = 3 \) and \( C_1 = 5 \) the actual used capacities will be \( u_{112}^{\text{actual}} = (\frac{1}{6})(1), u_{113}^{\text{actual}} = (\frac{1}{6})(3), u_{114}^{\text{actual}} = (\frac{1}{2})(3) \), summing to 5 units.

B. Distributed ILC Scheme

The theory described in the section II will now be used to minimize the variance around the nominal operating point found in III-A. Let \( \tilde{q} = q - q_{\text{nom}}, \tilde{u} = u - u_{\text{nom}} \) and let \( w \) be independent, zero-mean, Gaussian noise. Translating (7) around the nominal operating point \( q_{\text{nom}} \) we get:

\[
\tilde{q}[n+1] = \tilde{q}[n] + B\tilde{u}[n] + w[n] \quad (13)
\]

The matrix \( B \) will be on the form \( [B_1 \ldots B_{|S|}] \) where each \( B_j \) correspond to each server in \( S \). The column of each \( B_j \) consists of all column vectors \( [B_j]_{in, out} \), for \( (in, out) \in Q_j \times Q_j^\text{out} \), with zeros in all entries except for a 1 in position \( in \) and \( -1 \) in \( out \).

The servers are not allowed to use measurements of all queues to decide which of the queues to pick up from and deliver to. Each server \( s_j \) may only receive measurements from the queues in \( Q_j \), i.e. only from the queues it serves. This imposes the restriction on the feedback matrix \( L = \begin{bmatrix} L_1 & \ldots & L_{|S|} \end{bmatrix}^T \) where \( L_j \) is associated with server \( s_j \). The columns in each \( L_j \) must satisfy:

\[
[L_j]_{ii} = 0 \quad \text{when } i \notin Q_j
\]

With the structure imposed on \( L \) this means that \( BL = \sum_{j \in S} B_j \sum_{l \in Q_j} L_{jl} \) where the elements of each \( B_j L_j \) are:

\[
[B_j L_j]_{ik} = \sum_{m=1}^{|Q_j|} b_{im} l_{mk} = \sum_{(i,m) \in Q_j} \ell_{mk}
\]

Hence \( [B_j L_j]_{ik} = 0 \) unless \( k \in Q_j \). Now, we will use the theory presented in II for an algorithm to control the system of queues. The distributed algorithm becomes

Algorithm 1: Between times \( n_k \) and \( n_{k+1} - 1 \), let the state feedback law be \( \tilde{u}[n] = -L_k \tilde{q}[n] \) with the constant feedback matrix \( L[k] \). To update the feedback matrix \( L[k] \) in each server \( s_j \):

1) Let the system of queues run for times between \( n_k \) and \( n_{k+1} - 1 \).
2) Each queue \( i \) determines its corresponding \( \lambda_i[n] \) by communicating with neighboring queues according to (5).
3) Each server \( s_j \) receives measurements of all \( \tilde{q}_i[n] \) and \( \lambda_i[n] \) for \( i \in Q_j \).
4) Each server determines estimates of all \( E \tilde{u}_i \tilde{q}_j \) for \( i \in Q_j \) and \( E \lambda_i \tilde{q}_j \) for \( i_1, i_2 \in Q_j \).
5) Each server determines a descent direction \( G_j \) in which to update its feedback matrix \( L_j[k] \) by:

\[
G_{ji} = -2 R_j [E \tilde{u}_i \tilde{q}_j]_{\text{est}} + B_j^T [E \lambda_i \tilde{q}_j]_{\text{est}}
\]

6) In each server, for all \( i \in Q_j \), let:

\[
L^{(k+1)}_{ji} = L^{(k)}_{ji} + \gamma G_{ji}
\]

for some step length \( \gamma \).
7) Increase \( k \) by one and go to 1).
Remark 1. In step 5 in Algorithm 1 one might presume that the second term $B_j^T (E \lambda q_i)_{est}$ requires all $\lambda_i$ but by the sparse structure of $B_j$ it is easily seen that only $\lambda_i$ for $i \in Q_i$ is needed to form the product $B_j^T \lambda_i$.

IV. CASE STUDY

A. Model overview

We now consider a scenario where autonomous vehicles pick up ready-product palettes from queues in front of production lines and deliver them to stretch-wrapping stations for further processing. These stations lie across the queues as shown in Fig. 5 and process palettes at a very fast rate. Vehicles, are only limited to operating on a subset of the queues.

We assume that there exists a vehicle routing algorithm with time windows [2] which guarantees that vehicle $j$ can pick up at least $C_j$ palettes per unit time from any of its assigned queues and deliver them to its respective station.

A nonzero-mean random number $w_i[n]$ of palettes is generated at each queue $i$ at time period $n$, and each vehicle $j$ picks up $u_{jk}[n]$ palettes from queue $k$ at the same time (the third subscript index is not needed here).

The goal is to use our scheme to keep the number of palettes in each queue from varying above their capacity, as this requires halting of the production line and manual emptying of the queues, a very costly process.

B. Numerical Example

For a numerical example, we consider 4 production queues, each holding a maximum of 20 items. If a queue reaches this maximum, the production to that queue is stopped and restarted after the queues are manually emptied, something which is quite costly to the process. Note that this procedure is not shown in the simulation plots where we allow the queues to overflow for demonstration purposes.

Each queue is driven by the production noise $w \in \mathcal{N}(6,2)$. The queues are served by 6 servers, where each server has a capacity of 5 units per time sample.

The servers are allocated to the queues according to: Queue $i$ is served by server $i$, $i+1$ and $i+2$. With this scheme the sets $S_i$ become

$$S_i = \{ i, i+1, i+2 \} \quad 1 \leq i \leq 4$$

and sets $Q_j$ are

$$Q_1 = \{ 1 \} \quad Q_2 = \{ 1, 2 \} \quad Q_3 = \{ 1, 2, 3 \} \quad Q_4 = \{ 1, 2, 3, 4 \}$$

With this allocation setup and the knowledge of the noise statistic of the production rate, the nominal operating point for the servers can be determined using the method described in section III-A. The nominal operating points were only calculated once and are

$$\sigma_1 = \{ 3.5 \} \quad \sigma_2 = \{ 1.60, 2.46 \}$$

$$\sigma_3 = \{ 0.90, 1.77, 1.77 \} \quad \sigma_4 = \{ 1.77, 1.7, 0.90 \}$$

$$\sigma_5 = \{ 2.46, 1.60 \} \quad \sigma_6 = \{ 3.5 \}$$

When the nominal operating point is determined, the online iterative gradient descent method for ILC can be used to minimize the variance of the number of units on the queues.
The cost that will be considered is

\[
E \left( \sum_{1 \leq i \leq N_q} |\tilde{q}_i(t)|^2 + \sum_{1 \leq j \leq N_v} \left( \left| \sum_{i \in Q_j} \tilde{u}_{ij}(t) \right|^2 + \varepsilon |\tilde{u}_{jj}(t)|^2 \right) \right)
\]

where \( \varepsilon = 10^{-4} \) is to insure that the resulting weight matrix for the control signals, \( R \), is positive definite. The reason of summing the effort of each server over the queues it serves, is that the variance of the servers total effort is to be minimized.

Following the method in section III-B, the length of the time interval between updates of the feedback matrix is set to be 5 time samples (\( \Delta t_k = n_{k+1} - n_k = 5 \) for all \( k \)). The initial feedback matrix is set to be close zero, meaning that there is in principal no feedback in the beginning. The result of simulating the system for 300 time sample is shown in Fig. 6 together with the result of the same system simulated without feedback, i.e. only using the nominal control.

We see large improvement when introducing feedback. Worth noting is that the simulation with feedback actually starts without feedback, but before it can be seen in the plots, the gradient method is able to determine a feedback matrix which improves the performance. A large cost occurs when the non-feedback regulation goes above the queue capacity (here 20 items), where the production has to be stopped and the queues emptied.

We also compared our approach to a one where we used a longest-queue-first algorithm, i.e., the servers allocate their capacity so that the current largest residual from the reference is minimized. Although the results where quite similar, the computational time required by the longest-queue-first method was approximately 100 times larger than our method since it required the solution of a linear program at each time step.

V. CONCLUSION

We have outlined a comprehensive method for load balancing on queue networks, which using only local information regulates the variance of the networked buffer contents. It is a novel approach which uses a classical method from control engineering, namely LQG regulation, it is very fast computationally and distributed. With unbounded buffer and server capacity our method guarantees suboptimality bounds [11]. When the bounds are in place, our proposed quasi-static re-allocation of server bandwidths helps in reducing server overflow during variance regulation.

In future work we will look at the stability analysis of the quantized version of the scheme as well as stochastic analysis of the expected queue sizes. Currently the Q and R matrices are tuned in order to minimize overflow of the server capacities. For handling the server capacity constraints more rigorously we will also look at receding horizon approach, adapting work from [8].

REFERENCES