Report Number: WUCSE-2006-33

2006-01-01

Distributed Utilization Control for Real-time Clusters with Load Balancing

Yong Fu, Hongan Wang, Chenyang Lu, and Ramu S. Chandra

Recent years have seen rapid growth of online services that rely on large-scale server clusters to handle high volume of requests. Such clusters must adaptively control the CPU utilizations of many processors in order to maintain desired soft real-time performance and prevent system overload in face of unpredictable workloads. This paper presents DUC-LB, a novel distributed utilization control algorithm for cluster-based soft real-time applications. Compared to earlier works on utilization control, a distinguishing feature of DUC-LB is its capability to handle system dynamics caused by load balancing, which is a common and essential component of most clusters today. Simulation... **Read complete abstract on page 2.**

Distributed Utilization Control for Real-time Clusters with Load Balancing

Yong Fu, Hongan Wang, Chenyang Lu, and Ramu S. Chandra

Complete Abstract:

Recent years have seen rapid growth of online services that rely on large-scale server clusters to handle high volume of requests. Such clusters must adaptively control the CPU utilizations of many processors in order to maintain desired soft real-time performance and prevent system overload in face of unpredictable workloads. This paper presents DUC-LB, a novel distributed utilization control algorithm for cluster-based soft real-time applications. Compared to earlier works on utilization control, a distinguishing feature of DUC-LB is its capability to handle system dynamics caused by load balancing, which is a common and essential component of most clusters today. Simulation results and control-theoretic analysis demonstrate that DUC-LB can provide robust utilization control and effective load balancing in large-scale clusters.

2006-33

Distributed Utilization Control for Real-time Clusters with Load Balancing

Authors: Yong Fu, Hongan Wang, Chenyang Lu, Ramu S. Chandra

Abstract: Recent years have seen rapid growth of online services that rely on large-scale server clusters to handle high volume of requests. Such clusters must adaptively control the CPU utilizations of many processors in order to maintain desired soft real-time performance and prevent system overload in face of unpredictable workloads. This paper presents DUC-LB, a novel distributed utilization control algorithm for cluster-based soft real-time applications. Compared to earlier works on utilization control, a distinguishing feature of DUC-LB is its capability to handle system dynamics caused by load balancing, which is a common and essential component of most clusters today. Simulation results and control-theoretic analysis demonstrate that DUC-LB can provide robust utilization control and effective load balancing in large-scale clusters.

Type of Report: Other

theory.

The rest of this paper is organized as follows. Section 2 reviews previous work related to our research. Section 3 formalizes the problem. Section 4 presents the design and analysis of DUC-LB. Section 5 provides simulation results and Section 6 concludes the paper.

2. Related work

A multitude of load balancing approaches have been proposed for real-time distribute systems [3, 4, 22, 27, 30, 31]. While load balancing is an effective approach for handling overload in parts of the system, it can not provide overload protection when the workload exceeds the capacity of the entire cluster. Instead of proposing a new load balancing algorithm, we aim to develop a utilization control algorithm that enforces desired utilization bounds in a real-time cluster with load balancing. Specifically, we focus on a common load balancing approach called *diffusive load balancing (DLB)* [6–8, 17] in this work. We choose DLB because it is a dynamic and scalable load balancing approach which is particularly suitable for large real-time clusters (see Section 4.1).

A survey of feedback performance control in computing systems is presented in [1]. Several papers applied control theory to real-time scheduling and utilization control of single-processor systems [2, 12, 16, 24, 26, 37]. These algorithms are designed to control the performance of a single processor and hence are not applicable to real-time clusters.

Several recent papers proposed *centralized* utilization control algorithms [21, 25, 33] for distributed real-time systems. These algorithms can not scale effectively in large clusters. Among them the algorithm proposed in [33] also deal with distributed real-time system with load balancing but the effect of load balancing is not considered in the controller design. In contrast, both DFCS [28] and DEUCON [32] adopted distributed control approaches. However, neither of them can handle load balancing. As shown in our simulation results (Section 5), load balancing can have a significant impact on system stability under utilization control. A key novelty of our work is that we explicitly incorporate the dynamics of DLB in the model and design of our control algorithm. Another advantage of DUC-LB is that it only relies on admission control as the adaptation mechanism. In contrast, DEUCON uses task rates control, and DFCS assumes that tasks have multiple service levels. DUC-LB therefore can handle a more general workload model on real-time clusters.

A recent paper [13] proposed a control-based load balancing algorithm for optimizing the response times of a DB2 server through memory allocation. However, this algorithm uses a centralized controller and is not designed to control the CPU utilizations, which is important for a real-time cluster to deliver desired real-time performance.

3. Utilization Control in Real-time Clusters

In this section we first present the system model, and then formulate the utilization control problem in real-time clusters.

3.1. System Model

We assume a workload model that is a representative of soft real-time applications running on clusters such as real-time data streaming that periodically update stock prices or game scores. Thus we can model the workload as a set of periodic tasks with soft deadlines. A task T_i has a period p_i and an *estimated* execution time c_i. The estimated utilization of T_i, $U_i = c_i/p_i$. To avoid underutilizing the servers, these soft real-time applications are usually scheduled based on their estimated execution times (instead of worst-case execution times). However, the *actual* execution times are usually unknown. The utilization of the processor P_i, u_i, is the fraction of the time that the processor is not idle in a time interval. Note that the processor's utilization indicates the *actual* system load.

In a cluster, the processors are connected in a regular pattern by a high-speed network. The common topologies of regular network in distributed system include mesh, torus, cube and hyper cube [15]. We assume a network structure of *2-dimension torus* in this work, as shown in Figure 1. However, our analysis and design presented in this paper can be easily extended to other types of regular networks. A cluster comprises n^2 processors, $\{P_{i,j} \mid 1 \leq i, j \leq n\}$, which are *homogeneous* in terms of speed and functionality.

Each processor in the cluster executes the DLB algorithm for load balancing. DLB may migrate a task to a new processor after an instance of the task has been completed and before its next instance is started. After migration, the next instance will be started on the new processor. This mechanism is supported by many clusters. More details of DLB will be presented in Section 4.1.

3.2. Problem Formulation

Before formulating the problem, we first introduce some notations. The sampling period, T_s, of utilization control is assumed to be equal to the period of load balancing. We use $u_{i,j}(k)$ to denote the processor $P_{i,j}$'s utilization during the k^{th} sampling period $[(k-1)T_s, kT_s]$. The utilization set point of the processors, λ, is specified by the user based on the desired degree of overload protection and the schedulable utilization bound of the workload and the scheduling algorithm of the system. Because all processors are homogeneous and perform load balancing in the system model, we assume that all processors have the same utilization set point. For a processor $P_{i,j}$, the controller output, $r_{i,j}(k)$, is the change in the estimated utilization that need to be accommodated via admission control in the sampling period $[kT_s, (k+1)T_s]$. If $r_{i,j}(k) > 0$, the processor $P_{i,j}$ needs to admit some tasks, otherwise it needs to reject some tasks.

The utilization control in real-time clusters can be formulated as follows:

Problem. *Given the utilization set point λ for all processors in a real-time cluster with DLB, the goal of utilization control at the k^{th} sampling period is to choose $r_{i,j}(k)$ to minimize the following objective:*

$$\forall P_{i,j}, \quad \min \left(\rho_1 \left(u_{i,j}(k+1) - \lambda \right)^2 + \rho_2 r_{i,j}^2(k) \right), \quad (1)$$

where ρ_1, ρ_2 are optimization weights.

The optimization objective (1) includes two terms. The first term is the *tracking error*, i.e., the difference between the utilization and the set point. The system needs to minimize the tracking error in order to achieve the desired utilization. The second term is the *control cost.* A higher control cost requires the system to admit or reject more tasks which incurs higher run-time overhead. A user can achieve the desired balance between the tracking error and the control cost by assigning them appropriate weights.

4. Design and Analysis of DUC-LB

We present the design and analysis of DUC-LB in this section. As illustrated in Figure 1, DUC-LB employs a distributed control topology with a local controller located on each processor. Each local controller only communicates with its neighbors on the regular network. The primary goal of the control design is to guarantee global system stability and maintain utilization guarantees through localized control. In this section, we first present the dynamic model of real-time clusters with DLB as the foundation of our design. After introducing the key control variables we describe the localized feedback control loop in DUC-LB. We then present the detailed design and stability analysis of the control algorithm.

4.1. Dynamic Model of DLB

The fluid model suggested in [1, 18, 19, 29] is adopted to approximate the system dynamics. In the fluid model, a processor is approximated by a liquid vessel in which the level of liquid represents to the processor's utilization. The fluid model assumes that the utilization of each task is small when compared to the processor's capacity. This assumption holds in many clusters where each server executes a large number of tasks.

We now derive the dynamic model of a processor. Since a cluster is composed of homogeneous processors connected by a regular network, all processors share the same model. The global model of entire real-time cluster's dynamics can be captured by aggregating all the processors' model. The utilization of a individual processor in real-time clusters is influenced by two types of dynamics: the utilization variation of tasks running on the local processor and tasks migrated from or to neighbor processors due to load balancing.

Without considering load balancing, the utilization $u_{i,j}(k+1)$ of the processor $P_{i,j}$ can be modeled as [24]:

$$u_{i,j}(k+1) = u_{i,j}(k) + g_{i,j} r_{i,j}(k), \quad (2)$$

where $g_{i,j}$ is the *utilization gain* which represents the ratio between the *actual* utilization change and the *estimated* one.

A contribution of our work is a dynamic model for explicitly incorporating DLB. DLB makes load balancing decisions based on the local information in a distributed manner and migrates tasks between neighbors. Since DLB does not require any global information, it can scale to large clusters and retain the communication locality of the clusters. Specifically, DLB compares the utilization of the local processor, $P_{i,j}$, to its neighbors $\{P_{l,m} \in \mathcal{N}_{i,j}\}$, where $\mathcal{N}_{i,j}$ is the set of neighbor processors of $P_{i,j}$. When detecting a difference in the utilization between the local processor and its neighbors, DLB transfers tasks from a heavy load processor to a light load processor. According to the DLB algorithm, the dynamics of the processor $P_{i,j}$ under DLB can be modeled as follows:

$$u_{i,j}(k+1) = u_{i,j}(k) + \sum_{P_{l,m} \in \mathcal{N}_{i,j}} h_{i,j}^{l,m} \kappa_{i,j}^{l,m} \left(u_{l,m}(k) - u_{i,j}(k) \right), \quad (3)$$

where $\kappa_{i,j}^{l,m}$ is the *diffusive constant* and $h_{i,j}^{l,m}$ is the *utilization gain* of the tasks migrated between $P_{i,j}$ and $P_{l,m}$. $\kappa_{i,j}^{l,m}$ is a tunable parameter that determines the number of tasks that are migrated between $P_{i,j}$ and $P_{l,m}$ in response to the difference in their utilizations. Specifically, tasks with a total estimated utilization of $\kappa_{i,j}^{l,m} \left(u_{l,m}(k) - u_{i,j}(k) \right)$ are migrated from $P_{l,m}$ to $P_{i,j}$; on the other hand, when $u_{l,m} < u_{i,j}$, tasks with a total estimated utilization of $|\kappa_{i,j}^{l,m} \left(u_{l,m}(k) - u_{i,j}(k) \right)|$ are migrated from $P_{i,j}$ to $P_{l,m}$. $h_{i,j}^{l,m}$ is the ratio between the actual utilization of the tasks migrated between $P_{i,j}$ and $P_{l,m}$ and the estimated one. The DLB model (3) is subjected to two fundamental constraints [8]:

$$\kappa_{i,j}^{l,m} \geq 0, \quad 1 - \sum_{P_{l,m} \in \mathcal{N}_i} \kappa_{i,j}^{l,m} \geq 0. \quad (4)$$

To simplify the controller design we assume the *diffusive constant* on each processor share the same value κ so that a same controller can be applied to every processor in the cluster. Under this assumption, the constraints can be rewritten as more specific form, $0 \leq \kappa \leq 0.25$, because we restrict ourselves on *2-dimension torus*.

As depicted in Figure 1, $\mathcal{N}_{i,j}$, the set of neighbors of the processor $P_{i,j}$, can be identified directly as $\mathcal{N}_{i,j} = \{P_{i+1,j}, P_{i,j+1}, P_{i-1,j}, P_{i,j-1}\}$. Thus we can write the model of the processor $P_{i,j}$ in a real-time cluster by integrating the equation (2) and (3) into a specific form

$$u_{i,j}(k+1) = (1 - 4\kappa \sum_{P_{l,m}\in\mathcal{N}_{i,j}} h_{i,j}^{l,m})u_{i,j}(k) + \kappa \sum_{P_{l,m}\in\mathcal{N}_{i,j}} h_{i,j}^{l,m} u_{l,m}(k) + g_{i,j} r_{i,j}(k). \quad (5)$$

We call (5) as *nominal* model when $g_{i,j} = h_{i,j}^{l,m} = 1$, that is,

$$u_{i,j}(k+1) = (1-4\kappa)u_{i,j}(k) + \kappa\,(u_{i+1,j}(k) + u_{i-1,j}(k) + u_{i,j+1}(k) + u_{i,j-1}(k)) + g_{i,j} r_{i,j}(k). \quad (6)$$

The *nominal* model (6) captures the system dynamics in an ideal case under which there is no difference between *actual* and *estimated* utilization of the tasks.

4.2. Control Variables

Before describing the control design, we first introduce several important variables used in our control design. In this paper, we use $C_{i,j}$ to denote the controller located on the processor $P_{i,j}$.

- **Optimization variables:** Optimization variables are those that need to be minimized by the controllers. According to the problem formulation of utilization control (1), the optimization variables of the controller $C_{i,j}$ on the processor $P_{i,j}$ include the difference between the utilization and its set point as well as the change of estimated utilization. Formally,

  $$z_{i,j}^1(k) = u_{i,j}(k) - \lambda, \qquad z_{i,j}^2(k) = r_{i,j}(k). \quad (7)$$

  The first optimization variable is also the system output variable:

  $$y_{i,j}(k) = u_{i,j}(k) - \lambda. \quad (8)$$

  The second optimization variable, $r_i, j(k)$, is also the controller output, which is computed by the controller in the end of each sampling period.

- **Processor's interconnection variables:** Processor's interconnection variables represent the information exchanged between different processors for diffusive load balancing. There are two kinds of interconnection variables for each processor $P_{i,j}$: the *output* interconnection variable, which is the utilization of the local processor sent to its neighbors, and the *input* interconnection variable, which is a vector including the utilizations of neighbor processors received by the local processor. We denote the utilization of the processor $P_{l,m}$ received by the processor $P_{i,j}$ as $v_{i,j}^{l,m}(k)$ during the sampling period $[(k-1)T_s, kT_s]$. Likewise we denote the utilization of the processor $P_{i,j}$ sent to $P_{l,m}$ as $w_{i,j}^{l,m}(k)$. Accordingly, the processor $P_{i,j}$ has *input* and *output* interconnection variables, $\mathbf{v}_{i,j}(k)$ and $\mathbf{w}_{i,j}(k)$, respectively:

  $$\mathbf{v}_{i,j}(k) = \left[v_{i,j}^{i+1,j}(k), v_{i,j}^{i,j+1}(k), v_{i,j}^{i-1,j}(k), v_{i,j}^{i,j-1}(k)\right]^T,$$
  $$\mathbf{w}_{i,j}(k) = \left[w_{i,j}^{i+1,j}(k), w_{i,j}^{i,j+1}(k), w_{i,j}^{i-1,j}(k), w_{i,j}^{i,j-1}(k)\right]^T. \quad (9)$$

- **Controller's interconnection variables:** A key feature of DUB-LB is that it employs a neighborhood coordination scheme in which controllers exchange information with their neighbors in each sampling period. To represent the information exchange between the controllers, each controller also has *input* and *output* interconnection variables. We use $\mathbf{v}_{i,j}^K(k)$ and $\mathbf{w}_{i,j}^K(k)$ to represent the *input* and *output* interconnection variables of controller $C_{i,j}$, respectively. It is noted that the interconnection variables among *controllers* are related to but different from the interconnection variables among *processors*. We derive the interconnection variables among controllers as a key part of the controller design in Section 4.4.

4.3. Localized Feedback Control Loop

The architecture of a real-time cluster running DLB-UC is illustrated in Figure 1. In the end of each sampling period $[(k-1)T_s, kT_s]$, each controller performs the following steps.

1. *Information exchange among neighbors:* The local controller $C_{i,j}$ calculates and then sends its *output* interconnection variables $\mathbf{w}_{i,j}^K(k)$ to its neighbor controllers. The controller also receives the *input* interconnection variables $\mathbf{v}_{i,j}^K(k)$ from its neighbor controllers.

2. *Local Control computation:* Before the local controller $C_{i,j}$ computes its controller output, it needs two inputs. The first input is output variable, $y_{i,j}(k)$. The

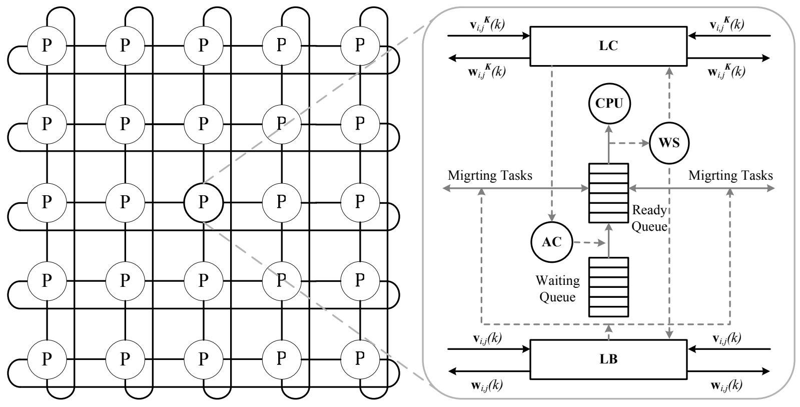


Figure 1. DUC-LB Framework Overview(5 $\times$ 5 *torus*)

workload sensor (WS) measures utilization $u_{i,j}(k)$ in the last sampling period and computes the system output $y_{i,j}(k)$. The other input is the controller's *input* interconnection variables $\mathbf{v}_{i,j}^K(k)$ received from the neighbor controllers. The controller then executes the control algorithm and generates the controller output $r_{i,j}(k)$ for the actuator to adjust processor's utilization.

3. *Utilization actuation:* DUC-LB uses the admission controller (AC) as the actuator. The actuator adjusts the processor's utilization locally according to the controller output. There are two ways to implement the actuator. First, the actuator can be implemented to reject or admit tasks on the local processor based on the controller output independently from the load balancers. This implementation is suitable to real-time clusters which are equipped with existing load balancers that cannot be modified. The second implementation reduces the overhead caused by task migrations by combining the decisions of the controller and the load balancer. In this case the actuator admits or rejects tasks based on the outputs of both the local controller and the load balancer. For instance, the load balancer wants to request the neighbor processors migrating tasks to the local processor, while the local controller wants to reject tasks from the local processor. Under such scenario the actuator may effectively reject the incoming tasks by not sending the request for task migration to the neighbors.

DUC-LB assumes that the communication delay of the interconnection variables is negligible when compared to the sampling period. This assumption holds in many clusters as they usually use high-speed networks to connect the processors. In addition, more significant network delays can be modeled as non-ideal channel between processors [20] and incorporated in the DUC-LB framework.

4.4. Design of the Distributed Controller

In this work we apply a recently developed distributed control design method [11] to design the distributed controller. This method extends the classic H_∞ optimal controller design [36] which aims to derive a controller to minimize the impact of disturbance on the system.

We first rewrite the *nominal* processor model (6) to the form that can be handled in the distributed control framework. Because the utilization gains are unknown at design time, we design the controller based on the *nominal* model. We then provide stability analysis that proves the robustness of the controller against variations in the utilization gains.

To reduce the number of variables, we first define the *spatial shift operator* $\mathbf{S}$ as ([11])

$$\begin{aligned}&\mathbf{S_x} v_{i,j}^{i-1,j}(k) = v_{i+1,j}^{i,j}(k), \mathbf{S_x^{-1}} v_{i,j}^{i+1,j}(k) = v_{i-1,j}^{i,j}(k), \quad (10)\\ &\mathbf{S_y} v_{i,j}^{i,j-1}(k) = v_{i,j+1}^{i,j}(k), \mathbf{S_y^{-1}} v_{i,j}^{i,j+1}(k) = v_{i,j-1}^{i,j}(k).\end{aligned}$$

Through *spatial shift operator*, the neighbor processor's *output* interconnection variables can be represented by the local processor's *input* interconnection variables because the local processor's *output* interconnection variables simply equal its neighbor processors' *input* interconnection variables. Considering that the utilization of the local processor is just the *output* interconnection variables, we can characterize the relation between *output* and *input* interconnection variables as

$$\mathbf{w}_{i,j}(k) = \Delta_{\mathbf{S}} \mathbf{v}_{i,j}(k) = [1,1,1,1]^T u_{i,j}(k), \qquad (11)$$

where $\Delta_{\mathbf{S}} = \mathbf{diag}(\mathbf{S_x}, \mathbf{S_y}, \mathbf{S_x^{-1}}, \mathbf{S_y^{-1}})$.

Since the input interconnection variables of the processor $P_{i,j}$ are composed of the output interconnection variables of its neighbor, and the output interconnection variables of $P_{i,j}$ are $\mathbf{w}_{i,j}(k)$, we can rewrite the *nominal* model of the processor (6) to accommodate distributed control design as follows

$$\begin{cases} u_{i,j}(k+1) = (1-4\kappa)u_{i,j}(k) + \kappa[1,1,1,1]\mathbf{v}_{i,j}(k) \\ \qquad\qquad\quad + r_{i,j}(k) \\ \Delta_{\mathbf{S}}\mathbf{v}_{i,j}(k) = [1,1,1,1]^T u_{i,j}(k) \\ y_{i,j}(k) = u_{i,j}(k) - \lambda(k) \\ z^1_{i,j}(k) = u_{i,j}(k) - \lambda(k) \\ z^2_{i,j}(k) = r_{i,j}(k) \end{cases} . \tag{12}$$

The model (12) has the form of standard H_∞ control problems except for the interconnection variables. The distributed controller design method adopted in this work extends the classic solution of H_∞ problem to deal with these interconnection variables [9–11]. The core of this design method is a set of Linear Matrix Inequalities (LMIs) which are generated by the stability and performance requirements (equation (82)-(84) in [11]). The distributed optimal controller can be constructed based on the results of these LMIs. The procedure of distributed controller design has been built in Matlab Multidimensional system toolbox [9]. The details of the theory and design method of distributed control can be found in [9–11]. The controller derived by distributed control theory has the general form as follows,

$$\begin{bmatrix} x_{i,j}(k+1) \\ \Delta_{\mathbf{S}}\mathbf{v}^K_{i,j}(k) \\ r_{i,j}(k) \end{bmatrix} = \begin{bmatrix} A^K_{TT} & A^K_{TS} & B^K_T \\ A^K_{ST} & A^K_{SS} & B^K_S \\ C^K_T & C^k_S & D^K \end{bmatrix} \begin{bmatrix} x_{i,j}(k) \\ \mathbf{v}^K_{i,j}(k) \\ y_{i,j}(k) \end{bmatrix}, \tag{13}$$

where $x_{i,j}(k)$ is the state of the controller, and $\mathbf{v}^K(k)$ is the input interconnection variables of the controller. The interconnection variables of neighbor controllers also have the relationship: $\mathbf{w}^K_{i,j}(k) = \Delta_{\mathbf{S}}\mathbf{v}^K_{i,j}(k)$. Our controller computes the interconnection variables $\mathbf{v}^K_{i,j}(k)$ and $r_{i,j}(k)$ by solving Equation (13) based on the method presented in [10]. We note that the controller has the same interconnection structure as that of the load balancer, that is, both of them receive and send their interconnection variables from or to the neighbors. This coordination among neighbor controllers enable DUB-LB to achieve system stability despite the dynamics caused by load balancing.

4.5. Stability Analysis

In the context of DUC-LB, a stable system refers to one in which the utilization of every processor converges to the set point. According to design procedure of distributed control, if the LMIs representing performance and stability are *feasible*, the distributed controllers can guarantee stability. However because the DUC-LB controller is derived based on the *nominal* model in which assumes all utilization gains to be 1, we still need to prove system stability under uncertain utilization gains. To simplify our analysis, we assume that all processors in the real-time cluster share the same utilization gain, that is, $g_{i,j} = g$ and $h^{i+1,j}_{i,j} = h^{i-1,j}_{i,j} = h^{i,j+1}_{i,j} = h^{i,j-1}_{i,j} = h$. This is a reasonable assumption for many clusters composed homogenous processors that process similar service requests. Thus it is necessary to validate the stability of the controller for uncertain utilization gains.

What following is the main steps of the stability analysis of DUC-LB under uncertain utilization gains.

1. Derive the controller based on the *nominal* processor model (12) with a fixed diffusive constant κ according to the distributed control design method presented in Section 4.4.

2. Construct the closed-loop system model by combining the controller derived in Step 1 and the *nominal* processor model. We denote this closed-loop system as the *nominal* closed-loop system.

3. Substitute the unitary utilization gains in *nominal* closed-loop model derived in Step 2 to the *actual* utilization gains g and h. We denote such closed-loop system as the *actual* closed-loop system.

4. Finally, derive stability region of the *actual* closed-loop system under the diffusive constant κ, that is, the region of varied utilization gains g, h in which closed-loop system is stable.

We present the main result of our stability analysis in the following theorem.

Theorem 1. *Under a fixed diffusive constant κ, the DUC-LB with varied utilization gains, h and g, is stable whenever $\theta_1 \le h \le \theta_2$ and $\gamma_1 \le g \le \gamma_2$ if the four corner systems which have parameters ($h = \theta_1,\ g = \gamma_1$), ($h = \theta_2,\ g = \gamma_1$), ($h = \theta_2,\ g = \gamma_2$), and ($h = \theta_1,\ g = \gamma_2$) satisfy the LMI conditions given in [10] for some* fixed *values of the scaling matrices.*

Note that this is a sufficient but not necessary condition. In the following we only sketch the key idea of the proof of Theorem 1 due to the space limit. We observe that the LMI conditions, which ensures the stability in controller design, depend linearly on g and h. Therefore, by continuity of the eigenvalues of matrices, if the stability conditions of *nominal* closed-loop system is derived, such conditions are also hold in a neighborhood of of the *nominal* closed-loop system. Theorem 1 provides a way to explore the stability region of DUC-LB only through probing four points in the region. Because both utilization gains, g and h, in DUC-LB are determined by the *actual* execution time of the tasks, we can use this result to validate the stability of DUC-LB when deviation between *actual* and *estimated* execution time occurs. As an example of application of Theorem 1, when the DUC-LB is designed based on the diffusive constant, $\kappa = 0.2$, a numerical computation in Matlab shows that stability holds for $0.8 \leq g, h \leq 1.1$. We apply the stability analysis to a more complex example in Section 5.3.

5. Performance Evaluation

In this section we first describe the simulation setup and and baseline algorithm for performance comparison. We then evaluate the performance and stability of DUC-LB under a wide range of dynamic workloads.

5.1. Simulation Setup

Our simulation environment consists of a real-time cluster simulator and a set of distributed controllers. The real-time cluster simulator is implemented in C++ based on RTSim [5] which is a simulation library for real-time systems. The real-time cluster simulator implements a set of interconnected processors each composed of a load balancer, a workload sensor, an admission controller, a scheduler, and a workload generator. The tasks on each processor are scheduled by the EDF algorithm [23]. The distributed controllers are implemented in Matlab using the Multidimensional system toolbox [9]. The real-time cluster simulator and the distributed controller communicate through Matlab APIs.

At the beginning of the simulation, the real-time cluster simulator initializes the distributed controllers. At the end of each sampling period, the simulator collects the processors' utilization and then calls the controllers. Each controller $C_{i,j}$ sends its output interconnection variable to its neighbor controllers, receives the *input* interconnection variables $\mathbf{v}_{i,j}^{K}$ from neighbor controllers, and executes the control algorithm. The controller also returns its output $r_{i,j}(k)$ to the real-time cluster simulator which in turn calls the admission controller to adjust the processors' utilization.

In our experiments, every task has a period of 2000 ticks (the time unit of the simulations) and an *estimated* utilization of 0.02, although their actual utilization may be different from the estimated one and vary at run time. Each task has a relatively small utilization to simulate real-time clusters that handle high volume of requests. While utilization bound of EDF is 1.0 under ideal conditions [23], we use 0.8 as the set point to provide additional overload protection against workload variations.

In the following experiments, the real-time cluster consists of 25 processors connected by a network with a *2-dimension torus* topology, as shown in Figure 1. The load balancer on every processor runs the Receiver Initiated Diffusive (RID) [34] algorithm. With the RID algorithm, a processor with a lower utilization requests tasks from neighbors with higher utilizations. Compared to the Sender Initiated Diffusive (SID) algorithm, RID not only achieves higher performance but also scale more effectively [34]. We set the *transferring threshold* to 0.01, i.e., half of a task's estimated utilization. If the difference of utilization between two neighbor processors is less than 0.01, task migration will not be performed to avoid possible *overbalancing*. The load balancing and the control loop share the same sampling period of 20000 ticks. The optimization weights in the optimization objective (1) are $p_1 = 0.1$ and $p_2 = 1.0$. Except when explicitly mentioned otherwise, the diffusive constant $\kappa = 0.2$.

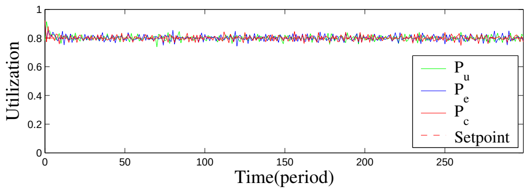
To evaluate the stability of DUC-LB, the average execution time of the task T_i can be tuned by execution-time factor $etf_i(k) = a_i(k)/c_i$, where $a_i(k)$ and c_i are the average and estimated execution times of task T_i, respectively. The actual execution times of the instances of the task are uniformly distributed in the range of $(0.8a_i(k), 1.2a_i(k))$. Unless mentioned otherwise, all the tasks on a processor share a same execution-time factor $etf(k)$. Thus the utilization gains, g and h, equal to $etf(k)$. During a simulation the execution-time factor can be either constant or changed dynamically.

5.2. Baseline Algorithm

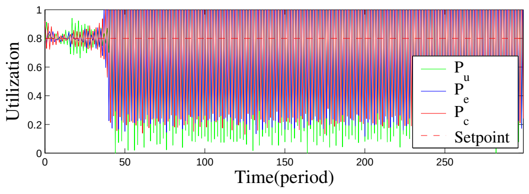
We use FCU-LB as the baseline for performance comparison. FCU-LB is a straightforward extension of a single-processor utilization control algorithm called FC-U algorithm [24]. With FCU-LB every processor has its own controller designed running the FC-U algorithm. Each controller computes its control output based only on the local processor's utilization, i.e., the controllers do not communicate with each other. FCU-LB may work well when task migration is rare, because under such scenario the system can be seen as an aggregation of isolated individual processors on which the performance of FC-U has been validated by previous research [24]. However, when task migration occurs frequently due to load balancing, the performance of FCU-LB may degrade sharply. Note that we can not compare DUC-LB against an existing distributed control algorithm such as DEUCON [32] and DFCS [28] because both of them have additional requirements on the task model. Specifically, DEUCON requires task rates to be adjustable, and DFCS assumes that tasks have multiple service levels. In contrast, DUC-LB supports a more general task model without those restrictions.

5.3. Steady Execution Times

This set experiments evaluate the performance of DUC-LB when task execution times deviate from estimation. Although all tasks share a fixed execution-time factor etf in each run, different processors still experience differ-



(a) DUC-LB



(b) FCU-LB

Figure 2. Utilization of DUC-LB and FCU-LB at $etf = 1.1$

ent utilizations because of randomized execution times. Figure 2(a) illustrates the utilization of three processors. Processors P_u, P_e and P_c are located on the upper-left corner, the upper-center, and the center of the real-time cluster, respectively. At the beginning of the execution, etf is set to 1.1, i.e., the average execution time of the tasks is 110% of their estimation. As a result all processors are over-utilized. DUC-LB dynamically adjusts a processor's utilization by rejecting tasks. After $10T_s$, the difference between the average utilization and the set point becomes less than 0.01, while the standard deviation of the utilizations of all processors is less than 0.03. As predicted in our stability analysis, the system maintains stable under $\kappa = 0.2$ and $etf = 1.1$. For comparison we also plot the utilization of FCU-LB under the same workloads in Figure 2(b). In sharp contrast to DUC-LB, FCU-LB causes the utilization to oscillate drastically resulting in an unstable system. This is because the controllers in FCU-LB are not designed to handle the dynamics caused by load balancing. This result demonstrates the importance of incorporating load balancing in the system model and design of utilization control algorithms.

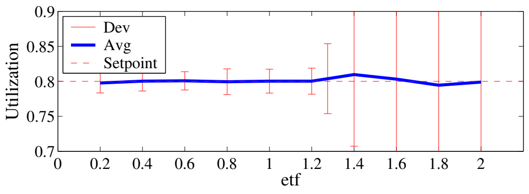


Figure 3. Average Utilization and Deviation of DUC-LB When Execution-time Factor Varies

To examine the performance of DUC-LB under different etf values, we plot the average and the standard deviation of the utilizations under DUC-LB when etf increases from 0.2 to 2.0 in Figure 3. Every data point in the figure is based on the utilization from $100T_s$ to $300T_s$ to exclude the transient response in the beginning of the run. As shown in Figure 3, the average utilization remains close to the set point for eft values from 0.2 to 1.3 and deviates from the set point for larger etf values. The deviation of DUC-LB shows the similar pattern. From $etf = 0.2$ to $etf = 1.28$ the deviation is less than 0.05 but increases significantly after that. If we consider the performance of DUC-LB acceptable when the deviation is less than 0.05, these results show that DUC-LB maintains desired utilization from $etf = 0.2$ to $etf = 1.28$. This range covers our analytical result that the system is stable between $etf = 0.8$ and $etf = 1.1$ at $\kappa = 0.2$. This is consistent with the claim that the analytical result is only sufficient but not necessary.

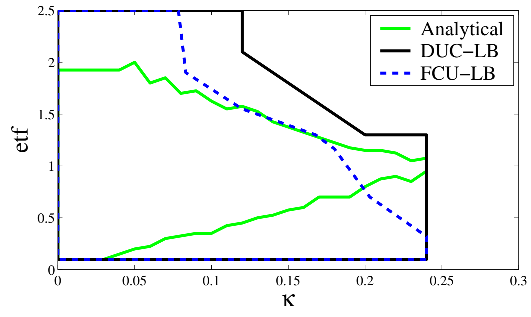


Figure 4. Stable Regions of DUC-LB, FCU-LB and Analytical Result

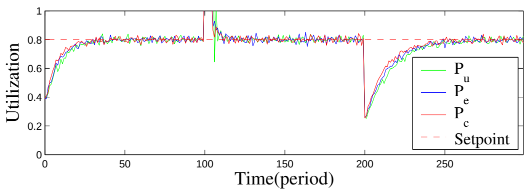
To further investigate the stability properties of the algorithms, we evaluate the stability of the DUC-LB under a range of parameter values, where $0.1 \leq etf \leq 2.5, 0.0 \leq \kappa \leq 0.24$. The range of the diffusive constant, κ, is based on the constraints of DLB (4). We still use deviation of the utilization as the criteria of stability, that is, the system is considered empirically stable if the standard deviation is within 0.05 and the average utilization deviates set point within 0.01. Figure 4 shows the empirical stable region of DUC-LB. For comparison we also plot the empirical stable region of FCU-LB, as well as the analytical stable region predicted by Theorem 1. Among these regions, the empirical stable region of DUC-LB is the largest. The analytical region of DUC-LB is inside the empirical stable region of DUC-LB. Again this is because the stability condition provided by Theorem 1 is sufficient but not necessary. The difference between FCU-LB and DUC-LB is particularly significant when both κ and etf are relative large. For example when $\kappa = 0.213$, which is the *optimal* diffusive constant of our real-time cluster's configuration [35], the stable range of FCU-LB is $0.1 \leq etf \leq 0.68$ while DUC-LB is

$0.1 \leq etf \leq 1.28$. The reason of poor performance of FCU-LB is that it ignores the interaction between the processors caused by load balancing.

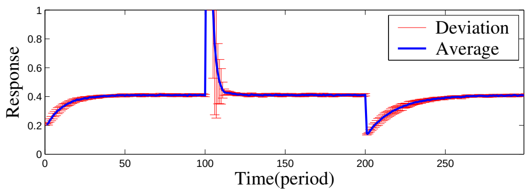
The different dynamic response of FUC-LB and DUB-LB can be illustrated with a simple example. Assuming a processor P has utilization 0.72 and each of its 4 neighbors have utilization of 0.85. The set point of utilization is 0.8. The FCU-LB *increases* utilization of P without considering the effect of load balancing because the measured utilization is less than the set point. But when κ is greater than 0.16, the tasks transferred from the neighbors via load balancing over-utilizes the processor P. In contrast, the controllers of DUC-LB exchange information with the neighbor controllers. Based on the exchanged information the DUC-LB controller is able to find that the local processor utilization will surpass the set point after load balancing and hence *decreases* the utilization of the local processor so as to maintain its desired utilization.

5.4. Varying Execution Times

In this experiment we demonstrate that DUC-LB can maintain desired utilization under varying execution times. We change etf dynamically according to the predefined profile in this experiment. In this profile, at the beginning of each run etf is 0.5; at $100T_s$ it increases abruptly to 0.9 which corresponds a 80% increase of execution time; at time $200T_s$ etf drops to 0.3 causing a 67% decrease in execution time.



(a) Utilization

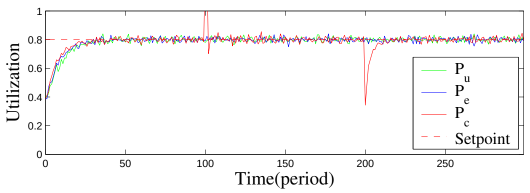


(b) Response Times

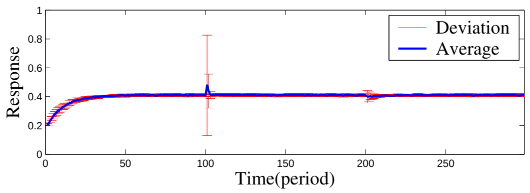
Figure 5. Utilization and Response Times under DUC-LB when execution time fluctuates globally

Figure 5(a) illustrates the utilization of DUC-LB. We observe that DUC-LB enforces the utilization set point even under significant load variations. At time $100T_s$, the increase in task execution times causes all processors to be overloaded. DUC-LB responds to the load increase by rejecting tasks and then the utilization returns to set point within $10T_s$. At time $200T_s$, etf drops, causing the system underutilized. DUC-LB responds by admitting more tasks and the utilization re-converges after a settling time of $60T_s$. The difference in the settling times after $100T_s$ and $200T_s$ are caused by the different utilization gains which change according to etf. The settling time after $100T_s$ is shorter because the utilization gains during interval $[100T_s, 200T_s]$ are larger than those in $[200T_s, 300T_s]$ and hence the system is more responsive to admission control after $100T_s$. Throughout the run DUC-LB achieves load balancing since the deviation of all processors' utilization is less than 0.02 when system is steady.

Figure 5(b) shows the average and the deviation of the Relative Response Time (RRT) of all processors in DUC-LB. RRT is defined as the ratio of the task's response time to its period. At the beginning of the run, because the execution time of the tasks is only a half of the estimated execution time, the RRT of the system remains a low level. Due to the load increase at $100T_s$, the RRT increases sharply. After DUC-LB causes the utilization to converge to the set point the RRT also decreases to a satisfactory level. At $200T_s$ the system load decreases 60% causing the RRT to drop. When DUC-LB adapts the utilization to the set point again the RRT also returns to the previous level. We also observe that the deviation of the RRT remains small in the process of adaptation due to the effect of load balancing.



(a) Utilization



(b) Response Times

Figure 6. Utilization and Response Times under DUC-LB when execution time fluctuates locally

The previous experiment represents a scenario with *global* load fluctuation. We now repeat the experiments under *local* load fluctuations. Only the tasks on the processor P_c (in the center of the cluster) change their execution times at run time in this experiment. Note that such local load fluctuations cause significantly imbalanced load in the system. As shown in Figure 6(a), DUC-LB still maintains desired utilization under *local* load fluctuations. The average response time, showned in Figure 6(b), only increases slightly when etf on P_e is varied. In addition the deviation of response time holds small in most running times except several short time intervals. These results demonstrate that DUC-LB can effectively handle the significant dynamics of load balancing caused by local fluctuations in system load.

6. Conclusions

We have presented DUC-LB, a novel distributed utilization algorithm designed for large-scale real-time clusters operating in dynamic environments. DUC-LB has several salient features. First, it is the first distributed utilization control algorithm that handles the dynamics of load balancing, which is a common feature of modern clusters. Second, it employs a localized control structure that can scale effectively in large clusters. Finally, DUC-LB is rigorously designed based on recent advance in distributed control theory. Stability analysis and simulation results demonstrate that DUC-LB can dynamically enforce desired utilization set points under a range of dynamic workloads.

7. Acknowledgements

Fu and Wang are supported in part by National Natural Science Foundation of China (Grant No. 60373055, 60542055 and 60374058). Lu is supported in part by an US National Science Foundation CAREER award (CNS-0448554). The authors would like to thank the anonymous reviewers for their valuable feedback.

References

[1] T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in software services. *IEEE Control Systems Magazine*, 23(3):74–90, June 2003.

[2] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole. Analysis of a reservation-based feedback scheduler. In *IEEE RTSS*, 2002.

[3] A. Amin, R. Ammar, and A. E. Dessouly. Scheduling real time parallel structure on cluster computing with possible processor failures. In *IEEE ISCC*, July 2004.

[4] R. Ammar and A. Alhamdan. Scheduling real time parallel structure on cluster computing. In *IEEE ISCC*, July 2002.

[5] C. Bartolini and G. Lipari. RTSim. http://rtsim.sssup.it/.

[6] J. E. Boillat. Load balancing and poisson equation in a graph. *Concurrency: Pract. Exper.*, 2(4):289–313, 1990.

[7] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Softw. Eng.*, 14(2):141–154, 1988.

[8] G. Cybenko. Dynamic load balancing for distriubted memory multiprocessors. *Journal of Parallel Distributed Computing*, 7:279–301, 1989.

[9] R. D'Andrea. Software for modeling, analysis, and control design for multidimensional systems. In *1999 CACSD*, pages 24–27, 1999.

[10] R. D'Andrea and R. S. Chandra. Control of spatially interconnected discrete time systems. In *IEEE CDC*, 2002.

[11] R. D'Andrea and G. E. Dullerud. Distributed control design for spatially interconnected systems. *IEEE Transactions on Automatic Control*, 48(9):1478–1495, Sept. 2003.

[12] D.C.Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A feedback-driven porpotional allocator for real-rate scheduling. In *OSDI*, Feb. 1999.

[13] Y. Diao, J. L. Hellerstein, A. J. Storm, M. Surendra, S. Lightstone, S. S. Parekh, and C. Garcia-Arellano. Incorporating cost of control into the design of a load balancing controller. In *RTAS*, 2004.

[14] G. E. Dullerud and F. Paganini. *A Course in Robust Conrol Theory, A Convex Approach*. Springer, 2000.

[15] T. Feng. A survey of interconnection networks. *IEEE Computer*, 14(12):12–27, 1981.

[16] A. Goel, J. Walpole, and M. Shor. Real-rate scheduling. In *IEEE RTAS*, 2004.

[17] A. Heirich and S. Taylor. A parabolic theory of load balance. Technical Report Caltech-CS-TR-93-25, California Institute of Technology, 1993.

[18] D. Henrich. The liquid model load balancing method. *Journal of Parallel Algorithms and Applications*, 8:295–307, 1996.

[19] C. C. Hui and S. T. Chanson. Hydrodynamic load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 10(11):1118–1137, 1999.

[20] C. Langbort, R. S. Chandra, and R. D'Andrea. Distributed control design for systems interconnected over an arbitrary graph. *IEEE Transactions on Automatic Control*, 49(9):1502–1519, 2004.

[21] S. Lin and G. Manimaran. Double-loop feedback-based scheduling approach for distributed real-time systems. In *HiPC 2003*, pages 268–278, 2003.

[22] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Real-time divsible load scheduling for cluster computing. Technical Report TR-UNL-CSE-2005-0011, University of Nabraska-Lincon, 2005.

[23] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time enviroment. *J. ACM*, 20:46–61, 1973.

[24] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Systems*, 23(1/2):85–126, July 2002.

[25] C. Lu, X. Wang, and X. Koutsoukos. Feedback utilization control in distributed real-time systems with End-to-End tasks. *IEEE Transactions on Parallel and Distributed Systems*, 16(6):550–561, June 2005.

[26] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware QoS management in web servers. In *IEEE RTSS*, 2003.

[27] K. G. Shin and C. J. Hou. Design and evaluation of effective load sharing in distributed real-time system. *IEEE Transactions on Parallel and Distributed Systems*, 5(7):704–719, 1994.

[28] J. A. Stankovic, T. He, T. Abdelzaher, M. Marley, G. Tao, S. Son, and C. Lu. Feedback control scheduling in distributed real-time systems. In *IEEE RTSS*, Dec. 2002.

[29] J. A. Stankovic, C. Lu, S. H. Son, and G. Tao. The case for feedback control real-time scheduling. In *ECRTS*, 1999.

[30] M. Suzuki, H. Kobayashi, N. Yamasaki, and Y. Anzai. A task migration scheme for high performance real-time cluster system. In *ISCA*, 2003.

[31] Y. C. Tay and H. Pang. Load sharing in distributed multimedia-on-demand systems. *IEEE Trans. on Knowledge and Data Engineering*, 12(3):410–428, 2000.

[32] X. Wang, C. Lu, D. Jia, and X. Koutsoukos. Decentralized utilization control in distributed real-time systems. In *IEEE RTSS*, Dec. 2005.

[33] Y. Wei, S. H. Son, J. A. Stankovic, and K.-D. Kang. Qos management in replicated real time databases. In *IEEE RTSS*, pages 86–97, 2003.

[34] M. H. Willebeek-LeMair and A. P. Reeves. Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(9):979–993, 1993.

[35] C. Xu and F. C. M. Lau. Optimal parameters for load balancing with the diffusion method in mesh networks. *Parallel Processing Letters*, 4:139–147, 1994.

[36] K. Zhou, J. C. Doyle, and K. Glover. *Robust and Optimal Control*. Upper Saddle River, NJ: Prentice-Hall, 1995.

[37] Y. Zhu and F. Mueller. Feedback EDF scheduling exploiting dynamic voltage scaling. In *IEEE RTAS*, 2004.

Appendix A. Proof of Theorem 1

We prove the main stability and performance result, Theorem 1, in a series of lemmas.

Lemma 1. *Let the closed-loop system, obtained by connecting the processor and controller, be represented by*

$$\begin{bmatrix} x^C_{i,j}(k+1) \\ (\mathbf{\Delta_S v}^C_{i,j})(k) \\ z_{i,j}(k) \end{bmatrix} = \begin{bmatrix} A^C_{TT} & A^C_{TS} & B^C_T \\ A^C_{ST} & A^C_{SS} & B^C_S \\ C^C_T & C^C_S & D^C \end{bmatrix} \begin{bmatrix} x^C_{ij}(k) \\ \mathbf{v}^C_{i,j}(k) \\ \lambda_{i,j}(k) \end{bmatrix}. \tag{A.1}$$

For the processor model (12) considered in this paper, the matrices A^C_{TT} and A^C_{TS} depend affinely *on the parameters h and g, while all other closed-loop realization matrices are independent of both h and g.*

Proof. The closed-loop realization matrices can be computed explictly as follows. Consider our processor model (12), which can be written as

$$\begin{bmatrix} u_{i,j}(k+1) \\ \mathbf{\Delta_S v}_{i,j}(k) \\ y_{i,j}(k) \\ z_{i,j}(k) \end{bmatrix} = \begin{bmatrix} A_{TT} & A_{TS} & B_{Tr} & B_{T\lambda} \\ A_{ST} & A_{SS} & B_{Sr} & B_{S\lambda} \\ C_{yT} & C_{yS} & D_{yr} & D_{y\lambda} \\ C_{zT} & C_{zS} & D_{zr} & D_{z\lambda} \end{bmatrix} \begin{bmatrix} u_{ij}(k) \\ \mathbf{v}_{i,j}(k) \\ r_{i,j}(k) \\ \lambda_{i,j}(k) \end{bmatrix}, \tag{A.2}$$

where $z_{i,j}(k) = [z^1_{i,j}(k),\ z^2_{i,j}(k)]^{\mathrm{T}}$, and the processor realization matrices are $A_{TT} = 1 - 4\kappa$, $A_{TS} = [\kappa,\ \kappa,\ \kappa,\ \kappa], B_{Tr} = 1, A_{ST} = [1,\ 1,\ 1,\ 1]^T, C_{yT} = 1, D_{y\lambda} = -1, C_{zT} = [1,\ 0]^T, D_{zr} = [0,\ 1]^T, D_{z\lambda} = [-1,\ 0]^T$, with $B_{T\lambda}$, A_{SS}, B_{Sr}, $B_{S\lambda}$, C_{yS}, D_{yr}, and C_{zS} being zero matrices of appropriate sizes.

From the controller model (13), the control output $r_{i,j}$ is

$$r_{i,j} = C^K_T x_{i,j} + C^K_S v^K_{i,j} + D^K y_{i,j}$$

Since D_{yr} and C_{yS} are zero matrices, we get

$$y_{i,j} = C_{yT} u_{i,j} + D_{y\lambda}\lambda_{i,j} \tag{A.3}$$

Thus,

$$r_{i,j} = C^K_{rT} x_{i,j} + C^K_{rS} v^K_{i,j} + D^K_{ry} C_{yT} u_{i,j} + D^K_{ry} D_{y\lambda}\lambda_{i,j}. \tag{A.4}$$

Substituting (A.4) in processor model (A.2) and (A.3) in controller model (12) and simplifying, we get (A.1), where the closed-loop state are $x^C_{i,j} := [u^T_{i,j},\ x^T_{i,j}]^T$, the closed-loop interconnection variables are $\mathbf{v}^C_{i,j} := [\mathbf{v}^T_{i,j},\ (\mathbf{v}^K_{i,j})^T]^T$, and the closed-loop realization matrices for this particular problem (where several of the processor matrices are zero) are

$$A^C_{TT} = \begin{bmatrix} A_{TT} + B_{Tr} D^K C_{yT} & B_{Tr} C^K_T \\ B^K_T C_{yT} & A^K_{TT} \end{bmatrix},$$

$$A^C_{TS} = \begin{bmatrix} A_{TS} & B_{Tr} C^K_S \\ 0 & A^K_{TS} \end{bmatrix},$$

$$B_T^C = \begin{bmatrix} B_{T\lambda} + B_{Tr} D_{y\lambda} \\ B_T^K D_{y\lambda} \end{bmatrix},$$

$$A_{ST}^C = \begin{bmatrix} A_{ST} & 0 \\ B_S^K C_{yT} & A_{ST}^K \end{bmatrix},$$

$$A_{SS}^C = \begin{bmatrix} 0 & 0 \\ 0 & A_{SS}^K \end{bmatrix},$$

$$B_S^C = \begin{bmatrix} 0 \\ B_S^K D_{y\lambda} \end{bmatrix},$$

$$C_T^C = \begin{bmatrix} C_{zT} + D_{zr} D^K C_{yT} & D_{zr} C_T^K \end{bmatrix},$$

$$C_S^C = \begin{bmatrix} 0 & D_{zr} C_S^K \end{bmatrix},$$

and

$$D^C = D_{z\lambda} + D_{zr} D^K D_{y\lambda}.$$

From the above explicit formulas, it can be seen at once that Lemma 1 holds. □

Now, define

$$\Gamma := \begin{bmatrix} A_{TT}^C & A_{TS}^C & B_T^C \\ A_{ST}^C & A_{SS}^C & B_S^C \\ C_T^C & C_S^C & D^C \end{bmatrix}$$

From the main result of [10], the stability and performance of the closed-loop system (A.1) are implied by the existence of symmetric *scaling matrices* $X_T > 0$ and X_S such that a matrix inequality of the form

$$J(X_T, X_S, \Gamma) < 0 \tag{A.5}$$

holds, where J is *affine* in A_{TT}^C and A_{TS}^C when X_T, X_S and the remaining elements of Γ are held fixed. From this observation and Lemma 1, we have the following result:

Lemma 2. *Assume that the controller (13) and the scaling matrices X_T and X_S are fixed. Then the matrix inequality $J(X_T, X_S, \Gamma) < 0$ which proves stability and performance of the closed-loop system is* affine *in h and g.*

Proof. Notice that, by hypothesis and Lemma 1, the only variable quantities that appear in the matrix inequality $J < 0$ are A_{TT}^C and A_{TS}^C. Since J is affine in A_{TT}^C and A_{TS}^C (when the scaling matrices X_T and X_S and the other elements of Γ are held fixed), the lemma follows immediately. □

Lemma 3. *(Nominal controller) Assume that for some nominal values of g and h, there exists a solution for the LMI* synthesis *conditions of [10]. Let the controller constructed by means of these LMIs be represented by (13). Then there exists an open set*

$$\mathcal{S} = \{(\alpha, \beta) : g_{\min} < \alpha < g_{\max},\ h_{\min} < \beta < h_{\max}\},$$

for some values of $g_{\min}$, $g_{\max}$, $h_{\min}$ and $h_{\max}$, such that the linear matrix inequality for stability and performance analysis (A.5) holds for some fixed scaling matrices X_T and X_S and for all $(g, h) \in \mathcal{S}$.

Proof. Since the LMI synthesis conditions in [10] are equivalent to (A.5), there exist, by hypothesis, scales matrices X_T and X_S that satisfy (A.5) for the *nominal* processor model and the corresponding controller. Now (A.5) depends affinely and hence continuously on h and g (from Lemma 2). Recall the meaning of the LMI (A.5): it is satisfied if and only if all eigenvalues of J are strictly negative. Since the eigenvalues of a matrix are continuous functions of its elements (see [14] for instance), (A.5) is also satisfied for small enough perturbations of h and g, which implies Lemma 3. □

The next lemma completes the proof by showing that the LMI (A.5) need only be checked at the vertices of the rectangle $\mathcal{S}$.

Lemma 4. *Assume that the* nominal *controller represented by (13) (which can be computed for the nominal processor according to the method given in [10]) satisfies the LMI (A.5) for some* fixed *values of $X_T > 0$ and X_S at the four vertices of the rectangle $\mathcal{S}$. Then in fact (A.5) holds for every point $(g, h) \in \mathcal{S}$.*

Proof. The proof is a simple special case of the convexity of linear matrix inequality conditions. Since both the controller and the scaling matrices are fixed, we have from Lemma 2 that the left hand side of (A.5) depends affinely on g and h. To make this dependence clear, we write the matrix function J as $J(p)$ for any point $p = (g, h) \in \mathcal{S}$. Define the four vertices of $\mathcal{S}$ as $p_1 := (g_{\min}, h_{\min})$, $p_2 := (g_{\max}, h_{\min})$, $p_3 := (g_{\min}, h_{\max})$, and $p_4 := (g_{\max}, h_{\max})$. By hypothesis, we have

$$J(p_i) < 0 \tag{A.6}$$

for $i = 1, \ldots, 4$. Since the rectangle $\mathcal{S}$ is convex, there exist real numbers α_i such that

$$\alpha_i \geq 0,\ i = 1, \ldots, 4 \tag{A.7}$$

$$\sum_{i=1}^{4} \alpha_i = 1, \text{ and} \tag{A.8}$$

$$p = \sum_{i=1}^{4} \alpha_i p_i \tag{A.9}$$

for all $p \in \mathcal{S}$. Since Lemma 2 implies that J is linear in g and h, we have

$$J(p) = \sum_{i=1}^{4} \alpha_i J(p_i)$$

for all $p \in \mathcal{S}$. Equations A.7 and A.8 imply that α_i are non-negative, and moreover, that α_i must be strictly positive for at least one value of $i \in \{1, \ldots, 4\}$. From this observation and (A.6), Lemma 4 follows. □