Feedback Control-based Dynamic Resource Management in Distributed Real-Time Systems

Tian He [**], John A. Stankovic [*], Michael Marley [*],
Chenyang Lu [***], Yin Lu [*****], Tarek Abdelzaher [****],
Sang Son [*], Gang Tao [*]

Department of Computer Science, University of Virginia
151 Engineer's Way, Charlottesville, Virginia 22904-4740

Abstract

The resource management in distributed real-time systems becomes increasingly unpredictable with the proliferation of data-driven applications. Therefore, it is inefficient to allocate the resources statically to handle a set of highly dynamic tasks whose resource requirements (e.g., execution time) are unknown a prior. In this paper, we build a distributed real-time system based on the control theory, focusing on the computational resource management. Specifically, this work makes three important contributions. First, it allows the designer to specify the desired temporal behavior of system adaptation, such as the speed of convergence. This is in contrast to previous literature, specifying only steady-state metrics, e.g. the deadline miss ratio. Second, unlike QoS optimization approaches, our solution meets performance guarantees with no accurate knowledge of task execution parameters – a key advantage in a poorly modeled environment. Last, in contrast to ad hoc algorithms based on intuition and testing, we rigorously prove that our approach not only has excellent steady state behavior, but also meets stability, overshoot, and settling time requirements.

Key words: Real-time, Feedback Control, Quality of Service, Scheduling

1 Introduction

Distributed real-time systems are widely used in highly dynamic environments where the resource requirements are open, fluctuating and not amenable to the traditional worst-case real-time analysis. For example, a web farm can be used to distribute time-sensitive contents such as movies and video clips. They need to handle a changing number of requests with significantly different resource requirements that are unknown beforehand. In a stock market, a system needs to actively push real-time stock updates at various interval to a group of users. The number of users served by a server can change quickly over time. Although these systems differ significantly in term of applications, they all operate in open environments where both workloads and available resources are difficult to predict. Monitoring and feedback control are needed to meet performance constraints. Several difficulties are observed in dynamic resource management in these systems. One main difficulty lies in their data-dependent resource requirements, which cannot be predicted without interpreting input data. For example, the execution time of an information server (a web or database server) heavily depends on the content of requests, such as the particular web page requested. A second major challenge is that these systems have highly uncertain arrival workloads; it is not clear how many users will request some resource in the web. A third challenge involves the complex interactions among many distributed sites, often across an environment with poor or unpredictable timing behavior. Consequently, developing certain types of future real-time systems will involve techniques for modeling the unpredictability of the environment, handling imprecise or incomplete knowledge, reacting to overload and unexpected failures (i.e., those not expressed by design-time failure hypotheses), and achieving the required performance levels and temporal behavior. We envision a trend in real-time computing to provide performance guaran-

* Dept of Computer Science, University of Virginia

** Dept of Computer Science & Engineering, University of Minnesota

* * * Dept of Computer Science & Engineering, Washington Uni. in St. Louis

* * ** Dept of Computer Science, University of Illinois, Urban-Champaign

* * * * * Dept of Computer Science & Engineering, University of Nebraska, Lincoln

tees without the requirement of fine-grained task execution models, such as those depending on the precise estimation of individual task execution times. We shall see the emergence of coarse-grained models that describe the aggregate behavior of resource requirements. Coarse-grained models are easier to obtain and they need not be accurately computed. These models are more appropriate for dynamic resource management in the presence of uncertainties regarding load and resources.

In this paper, we explore one such model based on difference equations. Unlike the more familiar queuing theory models of aggregate behavior, difference equation models do not make assumptions regarding the statistics of the load arrival process. Independent of the load assumptions, difference equation models are more suitable for systems where load statistics are difficult to obtain or where the load does not follow a distribution that is easy to handle analytically. The latter is the case, for example, with web traffic, which cannot be modeled by a Poisson distribution. Our solution has a basis in the theory and practice of feedback control scheduling. This is in contrast to the more common ad hoc resource management based on intuition and testing where it is very difficult to characterize the aggregate performance of the system and where major overloads and/or anomalous behavior can occur since these designs are not developed to avoid these problems.

2 The Overview of DFCS Architecture

Traditional real-time computing provides guarantees in avoidance of undesirable effects such as overload and deadline misses. They assume worst-case resource requirements known a priori. In contrast, in highly uncertain environments, the main concern is to design adaptation capabilities that handle uncertain effects dynamically and in an analytically predictable manner. To address this issue, we propose a framework called Distributed Feedback Control Real-time Scheduling (DFCS). The framework is based on feedback control that incrementally corrects system performance to achieve its target in the absence of initial load and resource assumptions. One main performance metric of such a system is the quality of performance-convergence to the desired level. In our framework, the desired convergence attributes may be specified and enforced using mechanisms borrowed from control theory. These mechanisms have been applied successfully for decades in physical process control systems that are often non-linear and subject to random external disturbances. Before establishing our DFCS framework, we give an overview of the software system being controlled and describe the feed-back-control mechanism involved. Note that although we focus on computational resource management here, while the general methodology can be applied to other dynamic resource management as well.

We assume that the resource under investigation is a cluster of computing nodes connected via a network. Tasks arrive at nodes in unknown patterns. Each task is served by a periodically invoked schedulable entity (such as a thread) with each instance having a soft deadline equal to its period. The periodicity constraint is motivated by the requirements of real-time applications such as process control and streaming media. We abstract a typical dynamic system by two sets of performance metrics. The *primary set* represents metrics to be maintained at specified levels, for example, the deadline miss ratio of a server, or the desired altitude of an airplane. The *secondary set* represents negotiable metrics such as service quality. The objective of adaptation is to incur minimum degradation in secondary metrics while maintaining the primary metrics at their desired values. To represent multiple levels of degradation in secondary metrics, we assume that each task has several service levels of different qualities. For example, a task can execute for varying amounts of time with the quality of the results improving with greater execution time. The goal of our DFCS architecture is to maintain the primary performance metrics around their desired values. Unlike a centralized system, the dynamics of a distributed system manifest themselves on two different time-scales. Fast dynamics are observed on individual nodes, while slower dynamics are observed on the entire system. The fast dynamics arise from local load changes due to individual task arrivals and terminations, while the slower dynamics arise from changes in aggregate load distribution. Therefore, our feed-back architecture naturally includes two sets of control loops,

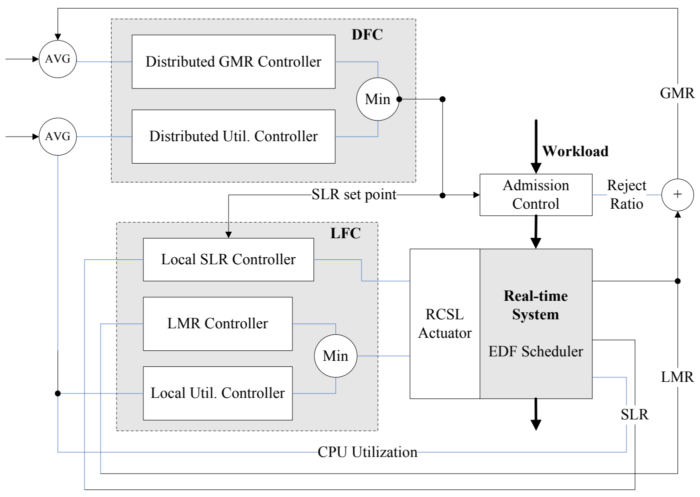

Fig. 1. DFCS Architecture Design

local and distributed ones, each tuned to the dynamics of the corresponding scale. Each node in the distributed system has a local feedback control system (LFC) and possibly a distributed feedback control system (DFC). The distributed feedback controller is responsible for maintaining the appropriate QoS balance between nodes. The local feedback controller is responsible for tracking the global QoS set point set by distributed controller and ensuring that tasks that are admitted to this node have a minimum miss ratio and the node remains fully utilized. It is important to note that these two types of controllers form the main parts of the distributed resource management in the system, but they are not the entire system.

Now consider a few more details about the DFCS architecture as shown in Figure 1. The distributed controller (DFC) commands a set of local controllers (LFC) via a QoS set point, termed as Service Level Ratio(SLR). The local controller (LFC) manipulates its actuators to achieve this SLR set point. In this architecture, we let the primary performance metric be the *deadline miss ratio* (MR). Since zero deadline miss ratio of admitted tasks can be trivially satisfied if the admitted task set is empty, it is especially important to quantify the loss of services due to task rejection to avoid trivial solutions. For this reason, we use two different miss ratio measurements, GMR and LMR. 1) GMR is the Global Miss Ratio of all submitted tasks, including both the admitted tasks and the rejected tasks. The distributed controller (DFC) is responsible for bounding the global miss ratio, GMR, of the system. 2) LMR is the Local Miss Ratio of admitted tasks in a single node. The local controller (LFC) is responsible for controlling the LMR miss ratio of locally admitted tasks as dictated by the distributed controller. Note that as shown in Figure 1 each of the local controller and the distributed controller has two similar parts, a miss ratio controller and a utilization controller. The miss ratio controller activates during overload, while the utilization controller activates at under-utilization when no deadline misses are observed, keeping the system sufficiently utilized. In addition, the LFC has a service level ratio controller (SLR) to address our secondary metric.

In addition, admission control (Figure 1) is based on estimated CPU utilization and the global SLR set point and decides to admit or reject tasks from the outside. If one task is rejected, it is offloaded to another node based on a certain routing policy. Finally, the real-time system is the *plant* under control, which processes the requests from the users. We can plug various scheduling algorithms into this real-time system based on different resource requirements. Here, we use EDF in our design.

3 Design and Model DFCS System

The DFC control design involves two components: a task model and difference equations describing the dynamics of the DFC in under-utilization and overload situations, respectively. The design process proceeds as follows: First, we specify the task model. Second, we specify the desired system performance using both transient and steady-state metrics. This step requires a mapping from the performance metrics of adaptive real-time systems to the dynamic response metrics of control systems used in control theory. Third we establish a mathematical model of the system for the purposes of feedback control. We take a high-level approach where our model aggregates the overall performance of the system in a single model. Our performance study shows this model works well, in spite of its simplicity. Finally, based on the performance specifications and the system model from first and second steps, we apply the mathematical techniques of control theory to design the controller that gives analytic guarantees on the desired transient and steady-state behavior at run-time. We map the controller design to various nodes in the system, depending on the network structure being studied.

3.1 Task Model

We assume a liquid task model in which a node can serve thousands of tasks, each with a small execution time. It is often the case in client-server architectures such as web servers. For each task T_i, there are N QoS service levels ($N \geq 2$). Task T_i running at Service Level q ($0 \leq q < N$) has a deadline $D_i[q]$ and an execution-time $C_i[q]$ that is *unknown* to the scheduler. The requested CPU utilization, $J_i(q) = C_i[q]/D_i[q]$, of the task is a monotonically increasing function of the service level q, which means that a higher QoS requires more CPU utilization. Let the average CPU utilization needed for a task set at level 0 be U_b. Without loss of generality, the average CPU utilization for a task set at level q is $f(q)U_b$, where $f(q)$ is a polynomial representing the Taylor's series expansion of the relation between CPU utilization and QoS level. Here we use the first order approximation of this relation to define the average requested CPU utilization $J(q)$ of a task set:$J(q) = (Aq+1)U_b$ where $q \in [0, N)$. Note that level $N-1$ is the best QoS a task can be served. It should be emphasized that the service level q of a single task T_i must be an integer value, however the service level q for a task set is the average service level, which can be a non-integer, if tasks are served at different levels in a single node. We make use of this approximation in the rest of the paper to derive the system model. Note that if this approximation is not appropriate in some situations, higher order ones can be used. However, the design process remains the same.

3.2 System Specification and Metrics

To evaluate the performance of a system, it is necessary to establish its specifications and performance metrics. Following the practice of the control community in specifying and evaluating the performance of control loops, we propose a series of canonic benchmarks that test system adaptation capabilities. These benchmarks generate a set of simple load profiles adapted from control theory; namely, the *step* load and the *ramp* load. The step load represents a worst-case load variation: a workload change that occurs in zero time. The ramp load represents a more moderate variation that features a slower rate of change. By experimenting with different rates of change, we can assess how well an adaptive system converges to the desired performance upon perturbations caused by changes in the workload. We can also analyze the effects of workload changes with different rates. If the change rate of the work is bounded, this analysis can yield guarantees on the convergence time and worst-case performance deviation. We measure the system load in the percentage of the full system capacity. The load corresponding to the full system capacity is said to be 100%. An overload is a system load that is higher than 100%. A load profile $L(t)$ is the system load as a function of time. In practice, this load is translated into system-specific parameters for evaluation purposes. For example, a 500% system load can be translated to the request rate of 8,000 Mbps in a specific web server.

Consider a time window $[(k-1)W, kW]$, where W is called the sampling period and k is called the sampling instant. During this window, let $M(k)$ be the number of task instances that miss their deadline, let $T(k)$ be the total number of task instances, and let $MR(k) = M(k)/T(k)$ be the miss ratio and M_S be the desired miss ratio performance, termed as the *set point* in control theory. To quantify the performance of adaptation, we have following metrics.

- **Overshoot** M_o**:** the maximum amount by which $M(k)$ exceeds its set point M_S, expressed as a percentage of the set point M_S.
- **Settling time** T_s: The time it takes the miss ratio to enter a steady state after a load change occurs.
- **Steady-state error** E_s: It is the difference between $M(k)$ and its set point M_S when no disturbance happens and after system transients have decayed. It indicates the DFCS's ability to regulate the controlled variable near the set point M_S in the long term. Ideally E_s should be zero.

These metrics provide a basis for us to compare the effectiveness of feedback control to other adaptive real-time scheduling policies. In addition, these metrics can be also used to specify the desired behavior of the adaptation process to guide the control loop design. To enforce these metrics, we need to establish a good aggregate model of the system, the central topic in the next section.

3.3 Modeling the Dynamics in DFCS

Before applying control theory to design a controller from specifications of adaptive behavior, it is necessary to model the system dynamics mathematically. Here the dynamics in DFCS is modeling as an integrated entity with aggregated behavior.

Let the utilization $U(k)$ be the fraction of time the CPU is busy in some sampling window k. Let $S(k)$ be the service level ratio (SLR) at the sampling window k, which can be formulated by Equation 1. In Equation 1, we assume each task has at least two service levels $(N \geq 2)$.

$$S(k) = \frac{AvgServiceLevel}{N-1} = \frac{\sum_{q=0}^{N-1} (Num.of\ Tasks\ completed\ at\ level\ q) \cdot q}{(Num.of\ Tasks\ completed) \cdot (N-1)} \quad S(k) \in [0,1]$$
$$S(k) = 0 \quad Num.of\ Tasks\ completed = 0 \tag{1}$$

Now we derive the relation between utilization $U(k)$, service level ratio $S(k)$ and the resulting number of miss $M(k)$. Note we use miss number $M(k)$ zero as desired the performance metrics (set point), which is equivalent to miss ratio $MR(k)$ zero. In each time window , CPU utilization is proportional to the number of tasks that finish successfully. This relationship can be modeled as:

$$U(k) = c(k) \cdot (T(k) - M(k)) \cdot J(q) \tag{2}$$

where $c(k)$ is the percentage of the arrived tasks that finish in the same sampling window. For example if $c(k) = 1$, all tasks arrive and finish in the same period. If $c(k) = 0.99$, 1% (relatively long) tasks neither miss their deadline, nor finish within the same period. From the perspective of control theory, worst-case conditions for convergence stability are those when system gain is maximum. The maximum gain condition corresponds to $c(k) = 1$. In other words, worst-case conditions occur when we assume the unfinished tasks consume their execution time in the sampling window of arrival. It is a reasonable assumption in a liquid task model where the task execution time is much smaller than the sampling window. Therefore, Equation 2 can be simplified as:

$$U(k) = (T(k) - M(k)) \cdot J(q) \tag{3}$$

From definitions of $J(q)$ in Section 3.1 and Equation 1, we have:

$$J(q) = (1 + A \cdot S(k) \cdot (N - 1)) \cdot U_b \tag{4}$$

Combining Equations 3 and 4, we get the relation desired:

$$U(k) = (T(k) - M(k)) \cdot (1 + A \cdot S(k) \cdot (N - 1)) \cdot U_b \tag{5}$$

Two important subcases arise in modeling the system: namely, overload and underutilization. They are modeled separately in the two subsequent subsections, respectively.

3.4 Modeling Dynamics when Overload

In the overload situation, tasks begin to miss their deadlines, there are two approaches to tackle the situation: admission control and service level ratio (SLR) adjustment. Admission control reduces a node's local miss ratio by rejecting incoming requests. SLR adjustment tries to accommodate more tasks by degrading the service levels of individual tasks. Since DFCS treats task equally, it degrades tasks with the highest service level first, until the average task service level ratio reaches the desired SLR. In the DFCS design, we deem task rejection the same as missing the task's deadline. Hence, we adopt SLR adjustment whenever possible. Here we get a difference equation that describes how SLR adjustment affects the number of misses when the system is overloaded ($M(k) > 0$). Since we assume the EDF scheduling, when deadline misses occur it must be that the CPU utilization $U(k)$ is 100%. We differentiate

Equation 5, setting $U(k) = 1$, we get the linearized small signal model of the system in overload situations:

$$\Delta M(k) = G_M \cdot \Delta S(k) + \Delta T(k)$$
$$where\ G_M = A(N-1)/(U_b(1 + A(N-1)S(k-1))(1 + A(N-1)S(k))) \tag{6}$$

3.5 Modeling Dynamics when Under-Utilization

The derivation in under-utilization situation is similar to Section 3.4. When DFC is underutilized under the EDF scheduling , the number of tasks missed deadlines $M(k)$ is obviously zero. Since the primary metric is satisfied, we focus on the SLR, the secondary metric presenting the QoS of admitted tasks. In this situation, we switch to utilization measurements. We can increase the SLR of the task set when the utilization is low to improve our service to the user. Here we obtain a difference equation that describes how SLR adjustment affects the CPU utilization when the system is underutilized ($U(k) < 100\%$). After we set $M(k) = 0$ and differentiate the Equation 5, we get

$$\Delta U(k) = G_U \cdot \Delta S(k) + G_t \cdot \Delta T(k)$$
$$where\ G_U = T(k-1)A(N-1)U_b\ \ and\ \ G_t = (1 + AS(k-1)(N-1)U_b \tag{7}$$

3.6 Design the Distributed Controller

With the DFCS dynamics models defined by Equation 6 and 7, we can now design the distributed feedback control loop. In this section, we first define the performance specifications to achieve, then we apply a control design method called Root Locus to tune the distributed controller. Due to space limitations, we do not review local controller design here, which has been intensely studied in our previous work [1] [2].

3.6.1 Design of the Control Loop

In the distributed case, each node in the DFCS provides the same SLR to the user. This property is often preferred in many distributed applications. For example, in a web server farm, the SLR of each HTTP request should be independent of where this request is served in the farm. So the major goal of the distributed controller is to calculate the SLR set point for the system. Since the system dynamics can be modeled with two difference Equations 6 and 7. The former describes the relation between the changes of the service level ratio and the changes of miss number when the whole system is overloaded; the latter models the relation between the changes of the service level ratio and the changes of CPU utilization when the system is underutilized. Based on this knowledge, we design the distributed feedback control loop. Because the external workload is not under our control, we deem $\Delta T(k)$ as the external disturbance[1] . Let G_U be the gain from $\Delta S(z)$ to $\Delta U(z)$ when the system is underutilized and G_M be the gain from $\Delta S(z)$ to $\Delta U(z)$ when system is overloaded. We get:

$$\begin{gathered} M(k) = M(k-1) + \Delta M(k) = M(k-1) + G_M \Delta S(k) \ when \ M(k-1) > 0 \\ U(k) = U(k-1) + \Delta U(k) = U(k-1) + G_U \Delta S(k) \quad when \ U(k-1) < 1 \end{gathered} \tag{8}$$

where G_M and G_U are defined in Equations 6 and 7, respectively. We can now draw the block diagrams of the feedback control system as shown in Figures 2 and 3. When the system is overloaded, the distributed miss feedback control loop (Figure 2)is activated. The components inside the dotted rectangle describe the dynamics of the controlled process with input $\Delta S(z)$ and output $M(k)$, where $C_M(z)$ is the miss controller to be designed and M_S is the miss set point. We can easily obtain the miss ratio $MR(k)$ by dividing $M(k)$ by the total number of tasks. Note that while the controlled system gain does change by a multiplicative factor if the output metric used is miss ratio $MR(k)$ instead, the overall loop gain remains the same. This is because the designed controller gain in this case is multiplied by the inverse of that

[1] It is possible to include external workload dynamics in the model by modeling the admission control process, however we found this is necessary only when the workload changes significantly over a very short period of time, which is not the case for most distributed system. Extension on this aspect is left as future work.

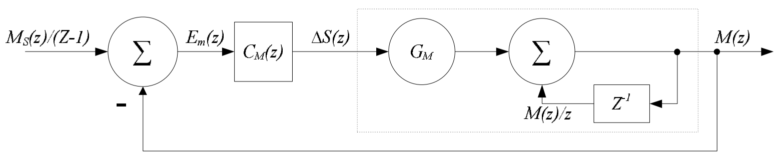

Fig. 2. Deadline Miss $M(k)$ Feedback Control Loop

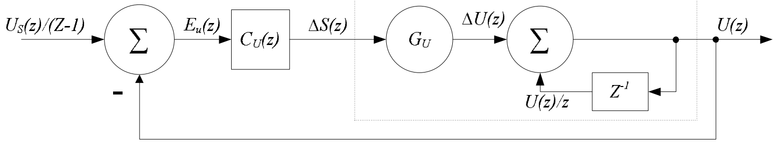

Fig. 3. CPU Utilization $U(k)$ Feedback Control Loop

factor. With the above observation in mind, the discussion below applies to both miss ratio $MR(k)$ and miss number $M(k)$ control. When the system is underutilized, we use the distributed utilization feedback control loop shown in Figure 3. $C_U(z)$ is the CPU utilization controller to be designed and U_S is the CPU utilization set point.

In z-transform notation we have:

$$\begin{aligned} M(z) &= H_M(z)\tfrac{M_S z}{z-1} \; \textit{where } H_M(z) = \tfrac{C_M(z)G_M}{1-z^{-1}+C_M(z)G_M} \\ U(z) &= H_U(z)\tfrac{U_S z}{z-1} \quad \textit{where } H_U(z) = \tfrac{C_U(z)G_U}{1-z^{-1}+C_U(z)G_U} \end{aligned} \tag{9}$$

Here the gains G_M and G_U are assumed to be set at some fixed values for nominal control design and analysis. Because our system intrinsically has an integral part, it is enough to use only a proportional controller to design $C_M(z)$ and $C_U(z)$ to guarantee the stability and zero steady state error. The general forms of the digital proportional controller in the time domain and z-domain are: $\Delta S(k) = K_p E(k)$(time domain)and $C(z) = K_P$ (z-domain). Here we denote K_M as the proportional term for the miss controller and K_U for the utilization controller. These values are substituted in Equations 9. Setting $C_M(z) = K_M$ and $C_U(z) = K_U$ we get:

$$\begin{aligned} M(z) &= H_M(z)\tfrac{M_S z}{z-1} \; \textit{where } H_M(z) = \tfrac{K_M G_M z}{(1+K_M G_M)z-1} \\ U(z) &= H_U(z)\tfrac{U_S z}{z-1} \quad \textit{where } H_U(z) = \tfrac{K_U G_U z}{(1+K_U G_U)z-1} \end{aligned} \tag{10}$$

3.6.2 Stability

According to control theory, system performance is determined by the poles of the closed loop transfer function. From Equations 10, we get $1/(1 + K_M G_M)$ as the pole for $H_M(z)$ and $1/(1 + K_U G_U)$ as the pole for $H_U(z)$. Since these poles are inside the unit cycle, according to the control theory, the stability is ensured in the DFCS system.

3.6.3 Steady State Error

Based on the Final-Value Theorem, the steady state values of and are:

$$\begin{aligned} \operatorname*{Lim}_{k\to\infty} M(k) &= \lim_{z\to1}(z-1)M(z) = \lim_{z\to1}\left\{(z-1)\tfrac{K_M G_M z}{(1+K_M G_M)z-1}\cdot\tfrac{M_s z}{z-1}\right\} = M_S \\ \operatorname*{Lim}_{k\to\infty} U(k) &= \lim_{z\to1}(z-1)U(z) = \lim_{z\to1}\left\{(z-1)\tfrac{K_U G_U z}{(1+K_U G_U)z-1}\cdot\tfrac{U_s z}{z-1}\right\} = U_S \end{aligned} \tag{11}$$

This result theoretically proves that the DFCS system can bring the miss number $M(k)$ and CPU utilization $U(k)$ to their set point(M_S and U_S)in steady state with zero error. It can also be verified that for a constant external disturbance $\Delta T(k) = \Delta T$, this asymptotic property still holds.

3.6.4 Settling Time

Settling time can be determined by the poles inside the unit cycle. The closer the pole is to the origin, the shorter the settling time. To deal with a worst case situation, we let $K_M G_M = 1$ and $K_U G_U = 1$, the poles of $H_M(z)$ and $H_M(z)$ are 0.5. According to control theory, the settling time is determined by the distance of the pole from the origin of the root locus plot. With a radius of 0.5, the theoretical settling time is about 8 sampling periods. In the experiment, based on the model, the calculated controller settings are 0.82 and 1.22 for the miss and utilization controllers, respectively.

3.7 Network Structures

For an effective distributed solution, we must consider the interaction among local controllers and the interaction between the global controller and local controllers. Two

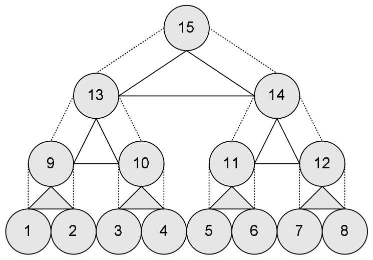

Fig. 4. Hierarchical Structure

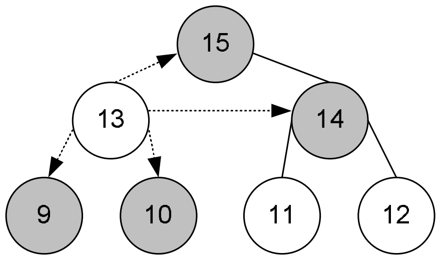

Fig. 5. Available routes in H-DFCS

aspects of the network to consider are the physical and logical network structure. The physical network structure could be a fully connected Ethernet, token ring, etc. For the purpose of this work, we are assuming that the physical structure of the network is either a hierarchy based point to point network, or a grid based point to point network running a Gigabit Ethernet.

The logical network structure defines information flow and route connectivity for the system. We propose two logical network structures: *hierarchical* and *neighborhood*, and design distributed real-time scheduling algorithm based on network structure.

3.7.1 Hierarchical Structure

Hierarchical Distributed Feedback Control System (H-DFCS) is based on the concept of information sharing in a hierarchical system. It allows a large distributed system to be broken down into a multi-level hierarchical system, as illustrated in Figure 4. By doing this, only the information that is required to coordinate subsystems needs to be exchanged at higher levels to coordinate the entire system. In the H-DFCS system, any node that has sub-nodes can be considered a coordinator. In the H-DFCS system, each node has a local feedback controller (LFC). The minimal requirement of this local controller is that it should be able to modify the service level set point of the node and report the local Miss Ratio (LMR) to other controllers (nodes).

The full scheduling algorithm for this system operates every sampling period in the following manner. Each node contains the LFC control system, with the exception of the top node in the hierarchy. This node contains the LFC control system as well as the DFC control system. The top node receives the MR and CPU utilization

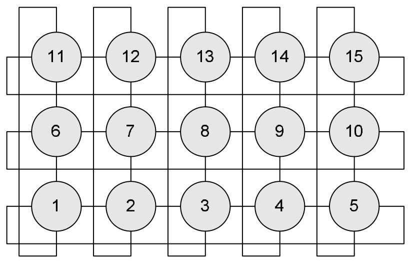

Fig. 6. Neighborhood Structure

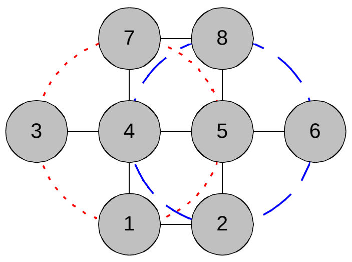

Fig. 7. Available routes in N-DFCS

averaged for the entire system and use this as inputs to its distributed controller to determine the new service level set point for the entire system. Such an operation allows the parent node to make decisions based on information from its children. The advantage is that the information from the children represents the subnet below that child, as the LMR values are weighted based on the number of nodes that the value represents.

Load balancing is achieved by migrating tasks between nodes through the network. Each node determines the route by comparing the MR values from the its children, parent and siblings. The MR values are weighted, to describe the number of nodes that they represent. This information is then used to assign a percentage to each entry in the route table, specifying the ratio of the off-loaded tasks to be sent along each route. In the hierarchical case that implements a binary tree, each node has up to four possible routes. For example, as shown in Figure 5, Node 13 has two routes to its two children, one to its parent, and one to its sibling. Routes are unidirectional, and are assigned only if the miss ratio of the other node in question is lower than that of the current node.

3.7.2 Neighborhood Structure

Neighborhood feedback control scheduling is based on the concept of information sharing in a neighborhood type system as shown in Figure 6. This means that a node shares its state information with its direct neighbors in the network and receives state information from these same neighbors.

Different from H-DFCS which has only one distributed controller at the root, in

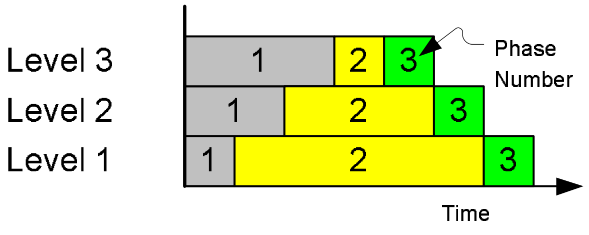

Fig. 8. H-DFCS in the 3-phase operation

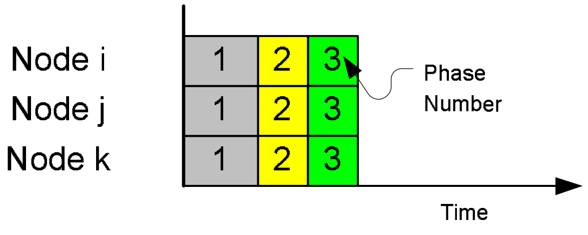

Fig. 9. N-DFCS in the 3-phase operation

neighborhood feedback control scheduling (N-DFCS) shown in Figure 6, every node contains both a local and distributed controller to control the state of the node as well as the state of its neighbor nodes. N-DFCS works within individual subnets instead of the whole distributed system. This design allows state information shared in a more decentralized manner. As a concrete example, Figure 7 shows the neighbors of two nodes (nodes 4 and 5) in a mesh structure (neighbors(4)=1,3,5,7 and neighbors(5)=2,4,6,8).

3.7.3 Comparison

Due to the difference in the network structure, H-DFCS and N-DFCS have different characteristics in terms of the delay and load balance.

- **H-DFCS:** Since H-DFCS needs to wait for the messages to be passed all the way up the hierarchy prior to making decisions and returning the set point, the duration of its operation depends on the number of levels in the hierarchy. For example, as shown in Figure 8, it takes seven time slots for a level-1 node finishes in a three-level hierarchy. This indicates that H-DFCS might not be quite suitable for some large scale systems. On the other hand, H-DFCS can make the SLR decision based on global information, leading to a quicker load balance.
- **N-DFCS:** N-DFCS allows a node to make a decision based on information from it nearest neighbors. The advantage here, is that state information is shared in a more decentralized manner. This helps in speeding up the computation time—as evident by comparing Figure 8 and Figure 9, but could result in a slower load balancing process. In comparison, the N-DFCS always works in a fixed time, regardless the number of nodes in the system.

4 Performance Evaluation

To evaluate the performance of DFCS, we compare N-DFCS and H-DFCS with two other well-known scheduling algorithms, QoS negotiation [3] and Dynamic QoS Management (DQM) [4]. Without a theoretical basis, QoS and DQM can be inefficient and difficult to tune. The results show that the feedback-control based approach is very effective. Due to space constraints, we are not able to show experimental results here. Please refer to [5] for complete evaluation results.

5 Related Work

Our DFCS architecture differs in two respects from early adaptive approaches. First, the performance of the adaptive system is modeled in a coarse-grained manner that represents the relation between aggregate QoS and aggregate resource consumption. This is different from fine-grained models, where the knowledge of individual task execution times is required as in [4]. Second, feedback control is used as a primary mechanism to adjust resource allocation in the absence of a priori knowledge of resource supply and demand. This is in contrast to early optimization-based QoS adaptation techniques that have assumed accurate models of application resource requirements. Examples of the new approach include [4,6]. In [6], a transaction scheduler called AED monitors the system deadline miss ratio and adjusts task priorities to improve the performance of EDF in overload. The DQM algorithm [4] features a feedback mechanism that changes task QoS levels according to the sampled CPU utilization or deadline misses. However, these algorithms are based on heuristics rather than a solid theoretical foundation.

Recently feedback control theory has been widely used as the underlying analytical foundation for building adaptive resource management systems. For example control theory has been applied to control throughput and delay [7] and to control congestion [8] at the network layer of the Internet. In addition, control-theoretic approaches have been adopted in a number of software systems such as realtime embedded systems [9,10], database servers [11], network storage systems [12]and web caching [2]. A survey on feedback performance control in software services is presented in [13]. Recent work on applying control-theoretic techniques in Real-Time systems is directly related to this work. In [14,1], feedback control real-time scheduling algorithms are developed to achieve deadline miss ratio guarantees in uniprocessor systems. Several recent paper also present feedback control algorithms (DEUCON [15]) and middleware (FC-ORB [16]) for enforcing desired utilizations of multiple processors in a distributed systems. There are several important differences between our work and those projects. First, those solutions control the resources by adapting the rates of end-to-end tasks. In contrast, our algorithms handle independent tasks via a combination of local QoS level adaptation and task (re)allocation.

6 Conclusions

To support data-driven applications with unpredictable and changing resource requirements, we develop here an effective computational resource management system, called DFCS, based on the feedback control. Different form other ad hoc approaches, DFCS has a basis in the theory. We have rigorously proven that our approach not only has excellent steady state behavior, but also meets stability, overshoot, and settling time requirements. We have demonstrated that DFCS is a better option for distributed resource management, than QoS [3] and DQM [4].

References

[1] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, Performance specifications and metrics for adaptive real-time systems, in: IEEE RTSS, 2000.

[2] Y. Lu, T. Abdelzaher, A. Sexana, Design, implementaion, and evaluation of differentiated caching services, accepted for IEEE Transactions on Parallel and Distributed Systems 15 (5).

[3] T. F. Adbelzaher, E. Atkins, K. Shin, QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control, IEEE Transactions on Computers 49 (11).

[4] S. Brandt, G. Nutt, T. Berk, J. Mankovich, A dynamic quality of service middleware agent for mediating application resource usage, in: RTSS '98, 1998.

[5] J. A. Stankovic, T. He, T. F. Abdelzaher, M. Marley, G. Tao, S. Son, C. Lu, Feedback Control Scheduling in Distributed Systems, in: IEEE RTSS, 2001.

[6] J. R. Haritsa, M. Livny, M. J. Carey, Earliest deadline scheduling for real-time database systems, in: IEEE Real-Time Systems Symposium, 1991, pp. 232–243.

[7] S. Keshav, A control-theoretic approach to flow control, in: Proceedings of the conference on Communications architecture & protocols, 1993, pp. 3–15.

[8] J. Wen, M. Arcak, A unifying passivity framework for network flow control (2002).

[9] L. Abeni, L. Palopoli, G. Lipari, J. Walpole, Analysis of a reservation-based feedback scheduler, in: RTSS '02, 2002.

[10] A. Cervin, J. Eker, B. Bernhardsson, , K.-E. Årzén, Feedback feedforward scheduling of control tasks, Real-Time Systems 23 (1-2) (2002) 25–53.

[11] S. Parekh, K. Rose, Y. Diao, V. Chang, J. Hellerstein, S. Lightstone, M. Huras, Throttling utilities in the ibm db2 universal database server, in: American Control Conference, 2004.

[12] M. Karlsson, C. T. Karamanolis, X. Zhu, Triage: performance isolation and differentiation for storage systems., in: IWQoS, 2004, pp. 67–74.

[13] T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, Y. Lu, Feedback performance control in software services, IEEE Control Systems Magazine 23 (3).

[14] C. Lu, J. A. Stankovic, G. Tao, S. H. Son, Design and evaluation of a feedback control edf scheduling algorithm., in: IEEE Real-Time Systems Symposium, 1999, pp. 56–67.

[15] X. Wang, D. Jia, C. Lu, X. Koutsoukos, Decentralized utilization control in distributed real-time systems, in: IEEE RTSS, 2005.

[16] X. Wang, C. Lu, X. Koutsoukos, Enhancing the robustness of distributed real-time middleware via end-to-end utilization control, in: IEEE RTSS, 2005.