Hierarchical Feedback Adaptation for Real Time Sensor-Based Distributed Applications

Mihaela Cardei, Ionut Cardei Rakesh Jha, Allalaghatta Pavan
Honeywell Technology Center
3660 Technology Drive, Minneapolis, MN 55418, USA
{mihaela, ionut}@cs.umn.edu, {jha, pavan}@htc.honeywell.com

Abstract

This paper presents an innovative hierarchical feedback adaptation method that efficiently controls the dynamic QoS behavior of real-time distributed data-flow applications, such as sensor-based data streams or mission-critical command and control applications. We applied this method in the context of the Real Time Adaptive Resource Management[1] *system, a middleware architecture for resource management with support for integrated services, developed at the Honeywell Technology Center. We present the analytical model for feedback adaptation for periodic distributed data-flow applications and we describe experimental results for an Automatic Target Recognition pipeline application and the impact of hierarchical feedback adaptation on the application behavior and its QoS parameters.*

1. Introduction

In recent years, there have been several efforts to build adaptive resource management systems for heterogeneous resources with real-time constraints [3,4,5,6,9,10]. This paper presents developments of the Real Time Adaptive Resource Management (RTARM) system [1,3], designed at the Honeywell Technology Center.

The RTARM system defines a hierarchical resource management architecture that provides the following basic services [1,3]: (1) scalable end-to-end criticality-based Quality of Service (QoS) contract negotiation that allows distributed applications to share common resources while maximizing their utilization and execution quality; (2) end-to-end QoS adaptation that dynamically adjust application resource utilization according to their availability while optimizing application QoS; (3) integrated services for CPU and network resources with end-to-end QoS guarantees and (4) real-time application QoS monitoring for integrated services. An innovative feature of RTARM is the hierarchical resource management architecture that unifies heterogeneous resources and their management functions into a uniform abstract resource model. In this paper, we refer to services and resources interchangeably. The central piece of the architecture is the Service Manager, a recursive structural component. This encapsulates a set of services and their management functions. Because all service managers export the same common interface, it becomes easy to build layered hierarchies recursively, in which heterogeneous services are integrated bottom-up. This also helps rapid object-oriented prototyping and development.

Many mission–critical distributed command and control applications, such as Automatic Target Recognition (ATR) [6], exhibit a degree of flexibility: they tolerate a range of QoS and resource usage above a minimum limit. Their performance depends on the allocated resources, usually specified by a contract, and they are ready to trade off some application service quality to save the critical services. For these applications, it is important to have a mechanism that regulates their dynamic behavior and protects them from contract violations.

The main contribution of this paper is a new hierarchical QoS-based real-time feedback adaptation method for periodic distributed data-flow applications with parallel-pipeline structure. We have developed an analytical model that enables control of the end-to-end QoS behavior for the entire distributed application by adjusting the input rate in the pipeline. This model can be generally applied to any type of periodic application with data-flow pipeline structure and a compatible QoS representation, such as multimedia streams and distributed command and control applications. We applied this model of feedback adaptation to our RTARM integrated service provider and experimented with a distributed ATR application.

Other adaptive real-time resource management systems are GRMS [4,5], ARA [10,11] and QualMan [9]. GRMS has a hierarchical structure that reflects the application data flow and does not offer feedback adaptation. The ARA framework [10] provides feedback adaptation for applications having a discrete set of acceptable

[1] Funded by DARPA under NRaD Contract number N66001-97-C-8524.

configurations with specific resource needs and accomplishes feedback adaptation by resource reallocation. [8] proposes a feedback adaptation method that adjusts the rate of data sent from a server to clients based on observation and prediction using a control-theoretical model. The system described in [7] uses digital control theory to determine the states of the adaptive system, which may activate control algorithms for adaptation. Another adaptive resource management system is QualMan [9], designed for distributed multimedia applications.

Our work differs from these approaches at the resource management architecture level, by supporting other application paradigms or by the way it accomplishes feedback adaptation.

The rest of this paper is organized as follows. In Section 2 we briefly describe the hierarchical architecture of the RTARM system. Section 3 presents the feedback adaptation model and analysis for the periodic parallel-pipeline applications. Section 4 continues with the description of the hierarchical feedback adaptation in RTARM, the ATR experiment, performance metrics and evaluation, and the impact of feedback adaptation on the ATR QoS. Section 5 concludes the paper and presents directions for future work.

2. The Real Time Adaptive Resource Management architecture

This section presents briefly the basics of the RTARM system. For more detailed information please refer to [1].

We have implemented an RTARM prototype that supports periodic independent tasks and periodic parallel pipeline applications with real-time requirements. The RTARM system is built as a middleware layer above the operating system and network resources. RTARM allows service initiation requests (admission requests) from clients and views applications as service consumers. RTARM supports a multidimensional representation of QoS, defined by a set of parameters (e.g. rate, latency, jitter) specified as a range [QoS_{min}, QoS_{max}]. The RTARM system strives to allocate the best available services to applications with priority for ones that are more critical.

2.1. Hierarchical adaptive service management for integrated services

The basic blocks of the RTARM recursive hierarchy, the Service Manager (SM), provide management functions for resources, such as CPU or network resources, and directly control resource utilization by application components. Higher-level service composition is based on lower-level services, giving rise to a service hierarchy. One use of a service hierarchy is to provide abstract or integrated resources for clients.

Figure 1 depicts a simple runtime configuration with two different lower-level SMs (LSM): a CPU and a Network SM at the bottom of the hierarchy, and a higher-level SM (HSM) that integrates CPU and network services. Two applications access services from the system.

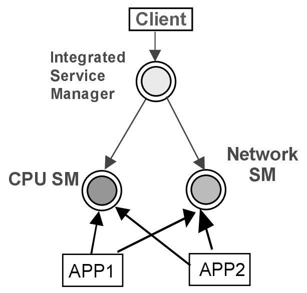

Figure 1. Sample RTARM hierarchy

2.2. Adaptation

RTARM recognizes three situations when application QoS may be changed after admission [1,3]: (1a) *QoS reduction* of lower criticality applications when a new application is admitted; (1b) *QoS expansion* when applications depart and release resources, and (2) *feedback adaptation*. While (1a) and (1b) imply contract changes and involve other applications, feedback adaptation does not change the contract, but only varies the current operational point of the application within the contracted QoS region. Feedback adaptation is triggered only by significant changes in application behavior, such as resource overload that results in a lowering of QoS operating point, resource underutilization that prompts RTARM to increase the application QoS operating point within the contracted QoS region and QoS contract violations that require corrective actions. Sections 3 and 4 detail feedback adaptation for pipeline applications.

2.3. Service Manager instances

We currently have implemented three service managers: CPU, Network and a higher-level Pipeline SM. All Service Managers have a component-based internal architecture with plug-and-play features [1].

2.3.1. CPU Service Manager. The CPU SM provides periodic applications access to a processor resource. Each computing node has a CPU SM, allowing concurrent applications to share a CPU. The application QoS is bidimensional: application execution rate (R) and iteration execution time (W). The specific CPU scheduling policy is isolated within the *Scheduler* component and the Monitor component tracks the CPU utilization. CPU feedback adaptation is presented in more detail in section 4.

2.3.2. Network Service Manager. We integrated the NetEx real-time network service manager [2,12] from Texas A&M University into the RTARM system by building a wrapper around it. NetEx runs as a middleware and provides connection-oriented real-time communication with guaranteed delay and bandwidth over commercial network infrastructures, such as ATM and switched Ethernet.

2.3.3. Pipeline Service Manager. The Pipeline Service Manager (PSM) is a higher-level SM that aggregates services from lower-level SMs (CPU, Network, other HSMs) into a representation suited for pipeline applications. A PSM client can be a user or another HSM.

Our PSM supports periodic independent tasks and periodic parallel pipeline applications, consisting of communicating stages in an arbitrary configuration, with a single source and a single sink node.

For periodic pipeline applications, the QoS consists of end-to-end message latency and rate for the final stage. The admission contract also contains execution time for each stage as well as the message size for each connection. It is the job of the pipeline *Translator* component, to decompose the integrated-service pipeline request into CPU and network admission requests. The admission protocol is described extensively in [1].

The PSM also provides hierarchical feedback adaptation that continuously monitors application QoS parameters and controls their resource utilization, taking corrective actions if necessary (refer to section 4).

3. Feedback adaptation model and analysis for pipeline applications

This section presents a model for periodic pipeline applications and introduces an efficient and stable method for feedback adaptation. We consider the end-to-end latency as the most critical QoS parameter. The main result is that by adjusting only the period for the input sensor, the system controls the end-to-end latency of a pipeline application. We further prove the convergence and stability of this algorithm. First, we list some assumptions.

A pipeline application consists of stage tasks that process data sequentially. We assume a sensor enters periodically data frames in the pipeline. Each frame is processed by each stage in turn and then sent to the next stage. A clock-based pipeline assumes that each stage operation is synchronous and periodic. If a frame is available for processing at the beginning of a period, the stage will process and send it to the next stage(s) in the data flow. If no frame is available at the beginning of a period, the stage will block until the beginning of the next period, when it will repeat the same cycle.

Our model ignores the network communication overhead between two stages. This assumption would not affect the feedback adaptation for the Automatic Target Recognition experiment because of the large disparity between the stage period (1-5s) and communication latency (0.05s).

The analysis assumes that the execution time and period of each stage are constant. These parameters may vary as the pipeline application evolves in time, and our analysis relates with a particular instance of time. It says that if starting with that moment the sensor input period is adjusted over some value, then, with the currently set parameters, the pipeline latency will exhibit deterministic behavior. In this way, the analysis may be applied at any time instance for the corresponding parameters.

Section 3.1 presents our main findings and an example for the clock-based simple pipeline. Section 3.2 generalizes for clock-based pipeline with composite stages.

We have also analyzed the event-driven pipeline model, which may be useful for other types of applications. This model assumes aperiodic stages. They may start execution of a frame whenever it becomes available. The results obtained for this model are similar to those of the clock-based model: the sensor input period is the only factor the pipeline application needs to adjust to control the pipeline end-to-end latency. Due to the space limitation, we do not describe this model here.

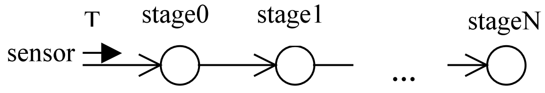

Figure 2. Linear, simple pipeline

3.1. Clock-based simple pipeline

Consider a pipeline with N+1 stages from Figure 2 and the next notations. N+1 is total number of stages, T is the period at which the sensor pushes frames into the pipeline. It may change over time, but we assume it stays constant starting with the frame with which we develop the analysis. C(i) is the execution (processing) time on stage i. T(i) is the period of stage i, $T(i) \geq C(i)$. W(i, n) is the waiting time for frame n at stage i. It represents the time the frame needs to wait before being processed by the stage i. It is greater than 0 if the stage i did not finish processing the previous frame. $W(i, n) \geq 0$ and $W(i, n) = \max [\, t_{out}(i, n-1) - t_{out}(i-1, n), 0 \,]$, where $t_{out}(i,n)$ is the time at which stage i produces output for frame n.

S(i, n) is the synchronization time. It is the time the frame n waits to synchronize with the beginning of the next period, for stage i. Always $0 \leq S(i, n) \leq T(i)$. $l(i, n)$ is the latency for frame n at stage i. $l(i, n) = C(i) + W(i, n) + S(i, n)$. $e(i, n)$ is the end-to-end latency up to and including the stage i, for frame n. $e(i, n) = \Sigma_{j=0..i}\ l(j, n)$

L(n) is the end-to-end latency for the whole pipeline, for frame n. $L(n) = e(N, n) = \Sigma_{j=0..N}\ l(j, n) = \Sigma_{j=0..N} C(j) + \Sigma_{j=0..N} W(j, n) + \Sigma_{j=0..N} S(j, n)$

Definition 1:
The pipeline is in the state S_k , where $0 \le k \le N$, for a frame x, if for all i = 0..k the relation (1) is true.

$$e(i, x) \le \Sigma_{j=0..i} (C(j) + T(j)) \qquad (1)$$

Observation: If a pipeline is in the state S_k, then it is also in states $S_{k-1}, S_{k-2}, S_{k-3}, \ldots, S_0$.

Definition 2:
We define the *stable region* for the end-to-end latency as the interval $[\ \Sigma_{j=0..N} C(j), \Sigma_{j=0..N} (C(j) + T(j))\ ]$.

We say the pipeline is in the stable region if its end-to-end latency is within the above interval.

If a pipeline is in the state S_N for frame x then it is in the stable region, because:

$$\Sigma_{j=0..N} C(j) \le L(x) \le \Sigma_{j=0..N} (C(j) + T(j))$$

The left limit for L(x) is evident, because $L(x) = \Sigma_{j=0..N} C(j) + \Sigma_{j=0..N} W(j, x) + \Sigma_{j=0..N} S(j, x)$ and $\Sigma_{j=0..N} W(j, x) \ge 0$, and $\Sigma_{j=0..N} S(j, x) \ge 0$.

From the application point of view it is important the pipeline latency be limited by an upper bound, because this guarantees it does not increase infinitely over time. The stable region of a pipeline corresponds to optimal pipeline behavior, in the sense that the end-to-end frame latency is bounded. We present next two theorems: the first refers to stability and the second handles the condition for convergence.

But before that, Lemma 1 proves a relation useful for the theorems' proofs.

Lemma 1:
If W(i, n) > 0 then the following relation is true:

$$e(i, n) = e(i, n\text{-}1) + T(i) - T \qquad (2)$$

Proof:
$W(i, n) > 0 \Rightarrow W(i, n) = t_{out}(i, n\text{-}1) - t_{out}(i\text{-}1, n)$
$t_{out}(i, n\text{-}1) > t_{out}(i\text{-}1, n) \Rightarrow S(i, n) = T(i) - C(i)$
$l(i, n) = C(i) + W(i, n) + S(i, n) =$
$=C(i) + W(i, n) + T(i) - C(i) = W(i, n) + T(i)$
$W(i, n) = t_{out}(i, n\text{-}1) - t_{sensor}(n\text{-}1) - t_{out}(i\text{-}1, n) + t_{sensor}(n\text{-}1)$
where $t_{sensor}(x)$ is the time instance when the sensor sends frame x.
$W(i, n) = e(i, n\text{-}1) - (t_{out}(i\text{-}1, n) - t_{sensor}(n\text{-}1) - T) - T = e(i, n\text{-}1) - e(i\text{-}1, n) - T$
$W(i, n) = e(i, n\text{-}1) - (e(i\text{-}1, n) + l(i, n)) + l(i, n) - T \Rightarrow$
$W(i, n) = e(i, n\text{-}1) - e(i, n) + l(i, n) - T$.
Implies $e(i, n) = e(i, n\text{-}1) + T(i) - T$.
$\therefore$

Theorem 1 refers to the case when pipeline is in the stable region. It proves that it is enough to maintain the sensor input period greater than the period of each stage in order to keep the pipeline in the stable region.

Theorem 1 (stability):
If the pipeline is in the stable region for frame n-1 and the sensor input period $T \ge max_{i=0..N} T(i)$, then the pipeline stays in the stable region for frame n.
Proof:
We show more generally, that if the pipeline is in the state S_k for a frame n-1, $0 \le k \le N$, and the input period $T \ge \max_{i=0..k} T(i)$, then the pipeline is in the state S_k for frame n.

We show by induction that $e(i, n) \le \Sigma_{j=0..i} (C(j) + T(j))$ $\forall$ i = 0..k
Step1: for i=0. We show that $e(0, n) \le C(0) + T(0)$).
We have one of the cases:
- $W(0, n) = 0 \Rightarrow S(0,n) \le T(0) \Rightarrow W(0, n) + S(0, n) + C(0) \le T(0) + C(0) \Rightarrow e(0, n) \le T(0) + C(0)$
- $W(0, n) > 0 \Rightarrow e(0,n) = e(0, n\text{-}1) + T(0) - T$ (use relation 2). $T \ge \max_{i=0..K} T(i) \Rightarrow T(0)\text{-}T \le 0 \Rightarrow e(0, n) \le e(0, n\text{-}1)$. The pipeline is in state S_k for frame n-1 $\Rightarrow$
$e(0, n\text{-}1) \le T(0) + C(0) \Rightarrow e(0, n) \le T(0) + C(0)$

Step2: suppose $e(i, n) \le \Sigma_{j=0..i}(T(j) + C(j))$, for i < k. We show that $e(i+1, n) \le \Sigma_{j=0..i+1}(T(j) + C(j))$
We have one of the cases:
- $W(i+1, n) = 0$. $S(i+1, n) \le T(i+1) \Rightarrow W(i+1, n)+S(i+1, n) + C(i+1) \le T(i+1) + C(i+1)$
We know that $e(i, n) \le \Sigma_{j=0..i}(T(j) + C(j))$. Implies $e(i+1,n) \le \Sigma_{j=0..i+1}(T(j) + C(j))$.
- $W(i+1,n) > 0 \Rightarrow e(i+1, n) = e(i+1, n\text{-}1) + T(i+1) - T$, (use relation 2). $T \ge \max_{j=0..K} T(j) \Rightarrow T(i+1) - T \le 0$, implies $e(i+1, n) \le e(i+1, n\text{-}1)$
The pipeline is in state S_k for frame n-1 $\Rightarrow$
$e(i+1, n\text{-}1) \le \Sigma_{j=0..i+1}(T(j) + C(j))$
Implies that $e(i+1, n) \le \Sigma_{j=0..i+1}(T(j) + C(j))$.
$\therefore$

Theorem 2 refers to the case when the pipeline is not in the stable region. It provides a solution to the case when the pipeline latency is too high, and it proves that it is enough to adjust the sensor input period in order to bring the pipeline end-to-end latency into the stable region, when the latency is superior limited.

Theorem 2 (convergence):
If the pipeline is not in the stable region for frame n-1 and starting with the frame n the sensor input period $T > max_{i=0..N} T(i)$, then the pipeline converges into the stable region after a finite number of frames.

Proof:

Let us note the pipeline current state I, where $I \neq S_N$. We show by induction that starting with frame n the pipeline behaves like:

$$\begin{array}{ccccccccc} I \rightarrow & S_0 & \rightarrow & S_1 & \rightarrow & S_2 \rightarrow & \ldots & \rightarrow & S_N \\ & m_0 & & m_1 & & m_2 & & & m_N \end{array}$$

where:

m_i is the number of frames needed by the pipeline in state S_{i-1} to converge in the state S_i, $0 \leq i \leq N$, $m_i \geq 0$, $\forall\, 0 \leq i \leq N$

Step1: show that $I \rightarrow S_0$ after a finite number of frames m_0. Suppose $I \neq S_0$ (otherwise we are done, with $m_0 = 0$). We show that for each new arriving frame x, e(0,x) decreases compared with previous frame value, until it becomes less than T(0) + C(0), at which time the pipeline is in state S_0. We have one of the cases:

- $W(0, x) = 0 \Rightarrow e(0, x) = W(0, x) + S(0, x) + C(0) \leq T(0) + C(0) \Rightarrow$ starting with this frame x the pipeline is in state S_0.
- $W(0, x) > 0 \Rightarrow e(0, x) = e(0, x-1) + T(0) - T$ (use relation 2). $T > \max_{i=0..N} T(i) \Rightarrow T(0) - T < 0$, implies that $e(0, x) < e(0, x-1) \Rightarrow$ end-to-end latency up to the stage 0 decreases between frames x-1 and x.

The same process happens again over successive frames, until the pipeline gets in the state S_0. The number of frames after which the pipeline gets in state S_0 is :

$$m_0 = \left\lceil \frac{e(0) - (T(0) + C(0))}{T - T(0)} \right\rceil$$

where e(0) is the end to end latency up to stage 0, when pipeline is in state I.

Note: the greater the input period T, the smaller m_o, so the earlier the pipeline converges to stage S_0.

Step2: Suppose the pipeline is in the state S_i. We show that after a finite number of frames, m_{i+1}, the pipeline enters state S_{i+1}. Suppose the pipeline is not in S_{i+1}(otherwise we are done with $m_{i+1} = 0$)

$\Rightarrow$ end to end latency up to the stage i+1 =

$= e(i+1) > \sum_{j=0..i+1}(T(j) + C(j))$

We show that for each new arriving frame x, e(i+1, x) decreases compared with previous frame value, until it becomes less than $\sum_{j=0..i+1}(T(j) + C(j))$, in which moment the pipeline is in state S_{i+1} .

We have one of the cases:

- $W(i+1, x) = 0 \Rightarrow e(i+1, x) = e(i, x) + W(i+1, x) + S(i+1, x) + C(i+1) \leq e(i, x) + T(i+1) + C(i+1)$

We know that $e(i, x) \leq \sum_{j=0..i}(T(j) + C(j)) \Rightarrow e(i+1, x) \leq \sum_{j=0..i+1}(T(j) + C(j))$

$\Rightarrow$ starting with this frame x the pipeline is in state S_{i+1}.

- $W(i+1, x) > 0 \Rightarrow e(i+1, x) = e(i+1, x-1) + T(i+1) - T$ (use relation 2). $T > \max_{j=0..N} T(j) \Rightarrow T(i+1) - T < 0 \Rightarrow$ $e(i+1, x) < e(i+1, x-1) \Rightarrow$ end to end delay up to the stage i+1 decreases between frames x-1 and x. The same behavior repeats over successive frames, until the pipeline gets in the state S_{i+1}. The number of frames after which the pipeline gets in state S_{i+1} is:

$$m_{i+1} = \left\lceil \frac{e(i+1) - \sum_{j=0}^{i+1} (T(j) + C(j))}{T - T(i+1)} \right\rceil$$

where e(i+1) is the pipeline end-to-end latency up to the stage i+1, at the instance the pipeline gets to state S_i.

Note: the greater the input period T, the smaller m_{i+1}, so the earlier the pipeline converges in stage S_{i+1}.

We have proved that by increasing the sensor input period above the maximum period of all pipeline stages, the end-to-end latency converges to the stable region. The theoretical results presented before proved the stability of our pipeline control method.

The next example illustrates how the pipeline end-to-end latency converges in time to the stable region when the input sensor period is set above the maximum period of all stages. Consider the following instance of a 9 stage pipeline in Figure 3:

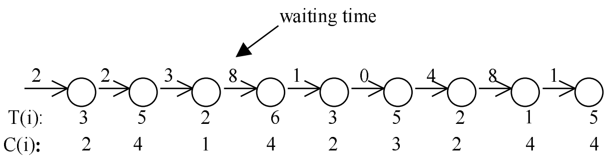

Figure 3. Example of linear pipeline

where T(i), C(i), T and the latency are represented in arbitrary time units. The end-to-end latency is 164, the stable region is [26, 68], and $\max_{i=0..8} T(i) = 11$. In conformity with Theorem 2, if the sensor input period is greater than 11, the end-to-end latency converges to the stable region. Figure 4 shows the pipeline behavior in time for T = 12, 13 and 14. We can observe that the greater the sensor input period T is, the earlier the pipeline enters the stable region. According to Theorem 1, once

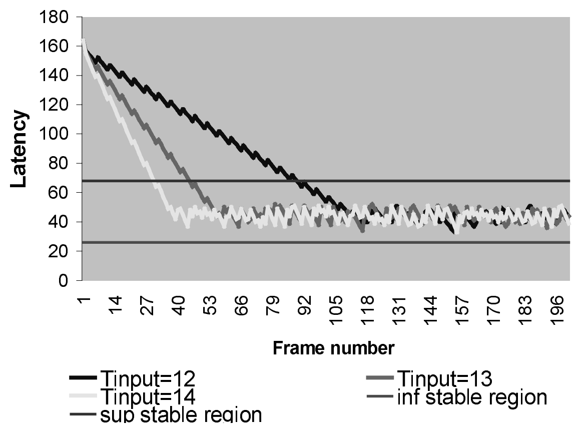

Figure 4. Latency variation depending on T_{input}

the pipeline enters the stable region, it remains there as long as $T \geq 11$.

3.2. Generalization for clock-based pipeline with composite stages

Some distributed data-flow applications have a complex structure with branches and parallel substages. One example is the ATR application depicted in Figure 5. We model these architectures as a linear pipeline with simple and composite stages. A simple stage represents a single, indivisible task that processes a frame. A composite stage i consists of substages arranged in parallel branches that process parts of a frame. A branch works like the simple linear pipeline presented in Section 3.1. When the last substage of each branch finishes the processing, the frame is reassembled at stage i+1.

We proved that for pipelines with composite stages the results obtained previously for clock-based simple pipeline are valid: setting the input period greater than the maximum period of all stages/substages guarantees the pipeline convergence to the stable region after a finite number of frames. Once it enters the stable region, the pipeline remains there as long as the sensor input period is greater than the maximum period of all stages/substages. Due to space limitation we do not present here the formal proofs.

4. RTARM hierarchical feedback adaptation for pipeline applications

The top-most HSM that receives the admission request directly from the client remains in control of the application QoS and its dynamics for its entire lifetime. That HSM is responsible for maintaining the distributed application's QoS within the contracted region and to improve it when possible using feedback adaptation. The resource management system must react quickly and adjust online the application parameters in case of allocated resource abuse or contract violation.

The RTARM hierarchy consists of a pipeline HSM, a network SM and several CPU SMs. The network SM does not provide feedback adaptation. The reserved network resources must cover the entire range of application rate. According to our analysis, it is possible to control the end-to-end frame latency for the entire pipeline just by controlling the rate of the input sensor or first stage. This allows the CPU SMs to conduct local feedback adaptation for each individual pipeline stage in order to provide locally the best QoS within the contracted range. Thus, feedback adaptation for the entire pipeline and CPU stages is conducted independently.

4.1. CPU Service Manager feedback adaptation

CPU SMs run pipeline stages just like any regular periodic independent task. In fact CPU SMs have no idea these tasks are part of a higher level entity, and they perform all RTARM functions in the same way. The CPU SM QoS consists of rate and iteration workload (execution time), both specified as intervals [min, max]. The CPU SM can directly control the application rate, but cannot touch the application workload. The CPU SM uses the product $CPU\_utilization = Rate \times Workload$ to asses schedulability. Applications send their actual QoS as events to CPU SM monitor at the end of each periodic iteration. The application is allocated a constant fraction L of the total processor time. At any time the current operational point (COP) may vary so that $R \times W \leq L$. The CPU SM adjusts the current operational point:

- increase rate when workload decreases
- decrease rate on overload

4.2. Pipeline feedback adaptation

The pipeline QoS parameter we consider critical and want to control is the end-to-end latency. As the pipeline evolves in time, rates of intermediate stages may change as a result of CPU SM feedback adaptation. In normal circumstances the input sensor period is maintained at a value greater than the period of any stage/substage of the parallel pipeline application, but it can get lower because of independent CPU feedback adaptation. When accumulation of queues between stages increases the end-to-end latency beyond a maximum threshold, the PSM sets the input sensor period at the maximum value from the pipeline contract. A finite state machine in the PSM maintains this maximal period for a fixed time, allowing the queues to empty. Then, the PSM sets again the input sensor rate to the maximal period of all stages. In this way, we know that the end-to-end latency decreases and after a finite number of frames the pipeline enters the stable region. This simple implementation allows the latency to oscillate within the stable region. A more sophisticated algorithm, topic for future research, would use target history and prediction to smooth the latency.

Our method is simple and efficient, as the only parameter to be adjusted is the sensor input period, while the pipeline stages are controlled only by the corresponding CPU SM. This mechanism avoids costly communication and coordination between the HSM and all the CPU SMs. The information required for pipeline feedback adaptation is minimal: the end-to-end latency for the current frame and the maximal period of all stages.

Another option for pipeline feedback adaptation would have been to let the PSM directly adjust online the rate for each stage. In this case the PSM would have to keep track of the current workload and rate, and maybe queue lengths for all stages, implying extra communication, processing overhead and lower resource utilization for CPU service managers.

4.3. The Automatic Target Recognition experiment

To test the feedback adaptation mechanism we designed a simple experiment with a true mission-critical application. The ATR pipeline, depicted in Figure 5, processes incoming video frames and displays detected targets. The sensor (stage 0) generates frames that pass through a sequence of filters and processing elements up to stage 6 that displays the original image and the targets. The 8-bit, 360x360 pixel, monochrome frames contain a variable number of targets (from 3 to 50). Stages 4, 5 and 6 expose a variable workload, proportional to the number of detected targets. Without pipeline feedback adaptation this would generate accumulations of frames in queues with latency increase.

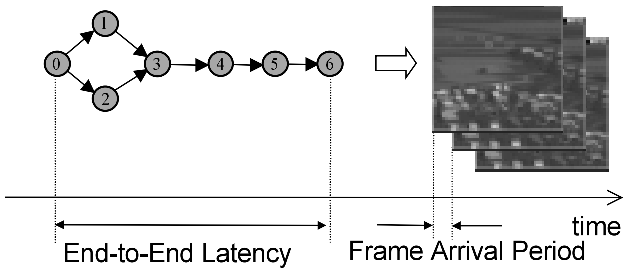

Figure 5. ATR pipeline application and QoS

4.4. Performance metrics and algorithm evaluation

We used three 450MHz NT Dell Workstation 400 machines, connected via a Fore ATM switch with OC-3c (155Mbps) links. Each machine runs a CPU SM. Both the network SM and the pipeline SM run on one of those three machines, and we consider their own CPU resource consumption negligible. Inter-SM CORBA signaling uses a secondary Fast Ethernet network, so that the ATM lines remain 100% available for the pipeline. For precise time measurements we used the NT performance counter.

The acceptable output frame period from the ATR pipeline contract is in [1,5]s, and the frame latency is [0.7,13]s. The seven ATR stages run at a variable workload within [0.02,1.5]s and within the same period interval [1,5] s.

We first present timing measurements for feedback adaptation at the CPU SM and PSM SM level. We measured the processing overhead of the feedback adaptation code and the time it takes the SM to react from the moment it receives the current QoS from the application until its adaptation command is enforced.

For CPU feedback adaptation, detection and enforcing the QoS adaptation takes around 4.4ms (Table 1), 3.9ms of the time being spent in a *set_qos()* operation, a two-way normal CORBA call. The pipeline adaptation enforcement includes a *set_qos()* call to the CPU SM that controls the sensor (or first stage) that calls directly the application with a *set_qos()* call. This explains why enacting pipeline QoS adaptation takes almost double the time than that for CPU SM QoS.

The actual rate and workload variation of stage 4 is shown in Figure 6. The stage has a variable workload and this causes the CPU SM to change the rate. Points A indicate overload that triggers rate decrease and points B indicate chronic underutilization that trigger rate increase.

Table 1. Feedback adaptation performance results for CPU SM and PSM

	Detection and decision processing	*Decision Enactment*	*Total Time*
CPU SM	0.508 ms	3.914 ms	4.422 ms
Pipeline SM	0.859 ms	6.816 ms	7.675 ms

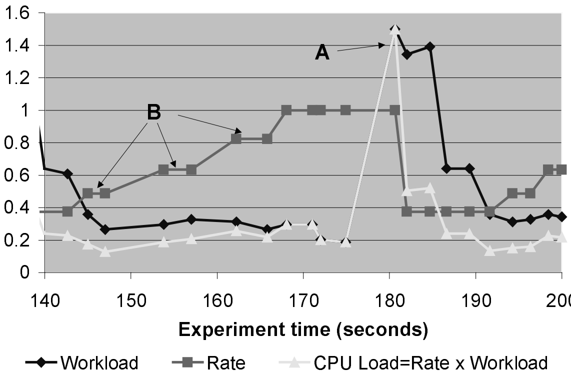

Figure 6. CPU SM feedback adaptation for a task with variable workload

The pipeline feedback adaptation mechanism makes sure the end-to-end latency and rate stay in the contracted range for the entire ATR pipeline (Figure 7). In order to practically demonstrate its effectiveness, we disabled the pipeline feedback adaptation after some time while keeping the sensor input period at a sustained low value of 1.48s (0.67Hz). Accumulation of frames in stage queues generated an increasing latency. A dotted line between points A and B marks the time when feedback adaptation was disabled. When the latency reached 30s, exceeding the maximal contracted value, we re-enabled pipeline feedback adaptation. The PSM sensor increased the sensor input period up at 5s (B). After a brief spike caused by the inertia of more than 23 frames already in transit through the pipeline, the latency went rapidly down (B → C), below the maximal threshold.

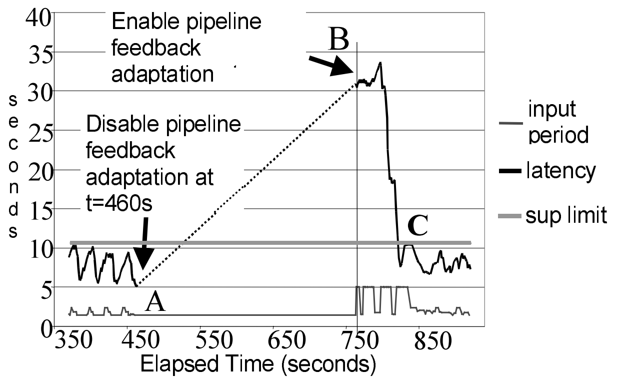

Figure 7. Latency variation for ATR with and without pipeline feedback adaptation

The preliminary ATR experiment shows that our hierarchical feedback adaptation algorithm proves to be effective and efficient. Detection, decision and enforcement at the pipeline level take less than 8ms and involve only the CPU SMs for the sensor stage and the last stage that actually reports the latency and rate. Still, further research would be necessary to fully assess the impact of the SM hierarchy on feedback adaptation performance.

5. Conclusion

This paper presented an innovative feedback adaptation mechanism for distributed data-flow applications with real-time requirements in the context of the RTARM project. We defined an analytical model and we proved its correctness and stability. We demonstrated its effectiveness by running an Automatic Target Recognition parallel pipeline application on a network of workstations managed by the RTARM system. Our hierarchical pipeline control method uses minimal information about the current state of the application and requires only one action to correct the end-to-end frame latency, while allowing feedback adaptation at the CPU level to function independently.

A direction for future work is to add prediction features to the current feedback adaptation method. Right now, it only takes corrective actions when the QoS falls below a threshold. Preventive actions would further decrease the overall pipeline reaction time. We also plan to study feedback adaptation for parallel pipeline applications where several pipeline HSMs have exclusive control over parts (sub-pipelines) of the entire distributed application.

References

[1] Cardei, I., Jha, R., Cardei, M., Pavan, A., "Hierarchical Architecture for Real-Time Adaptive Resource Management", to appear in Proceedings of the IFIP/ACM Middleware 2000 conference

[2] Devalla, B., Sahoo, A., Guan, Y., Li,C., Bettati, R., Zhao, W., "Adaptive Connection Admission Control for Mission Critical Real-Time Communication Networks", to appear in International Journal of Parallel and Distributed Systems and Networks, Special Issue On Network Architectures for End-to-end Quality-of-Service Support

[3] Huang, J., Jha, R., Heimerdinger, W., Muhammad, M., Lauzac, S., Kannikeswaran, B., Schwan, K., Zhao, W., Bettati, R., "RT-ARM: A Real-Time Adaptive Resource Management System for Distributed Mission-Critical Applications", Proceedings of the IEEE Workshop on Middleware for Distributed Real-Time Systems and Services, December 1997

[4] Huang, J., Wang, Y., Cao, F., "On Developing Distributed Multimedia Services for QoS and Criticality Based Resource Negotiation and Adaptation", Journal of Real-Time Systems, May 1999

[5] Huang, J., Wang, Y., Vaidyanathan, N.R., Cao, F., "GRMS: A Global Resource Management System for Distributed QoS and Criticality Support", Proceedings of the 4th IEEE International Conference on Multimedia Computing and Systems, June 1997

[6] Jha, R., Muhammad, M., Yalamanchili, S., Schwan, K., Rosu, D., deCastro, C., "Adaptive Resource Allocation for Embedded Parallel Applications", Proceedings of the 3rd International Conference on High Performance Computing, December 1996

[7] Li, B., Nahrstedt, K., "A Control Theoretical Model for Quality of Service Adaptations", Proceedings of Sixth International Workshop on Quality of Service, 1998

[8] Li, B., Xu, D., Nahrstedt, K., "Optimal State Predication for Feedback-Based QoS Adaptation", Proceedings of Seventh IEEE International Workshop on Quality of Service, 1999

[9] Nahrstedt, K., Chu, H., Narayan., S., "QoS-aware Resource Management for Distributed Multimedia Applications", to appear in Journal on High-Speed Networking, Special Issue on Multimedia Networking

[10] Rosu, D., Schwan, K., Yalamanchili, S., "FARA – A Framework for Adaptive Resource Allocation in Complex Real-Time Systems", Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium, June 1998

[11] Rosu, D., Schwan, K., Yalamanchili, S., Jha, R., "On Adaptive Resource Allocation for Complex Real-Time Applications", Proceedings of the IEEE Real-Time Systems Symposium, December 1997

[12] Sahoo, A., Li, C., Devalla, B., Zhao, W., "Design and Implementation of NetEx: A Toolkit for Delay Guaranteed Communications", Proceedings of Milcom, December 1997