Integrated Global and Local Quality-of-Service Adaptation in Distributed, Heterogeneous Systems

Larisa Rizvanovic[1], Damir Isovic[1], and Gerhard Fohler[2]

[1] Department of Computer Science and electronics, Mälardalen University, Sweden
{larisa.rizvanovic,damir.isovic}@mdh.se
http://www.mrtc.mdh.se
[2] Department of Electrical and Computer Engineering,University of Kaiserslautern, Germany
fohler@eit.uni-kl.de
http://www.eit.uni-kl.de/

Abstract. In this paper we have developed a method for an efficient Quality-of-Service provision and adaptation in dynamic, heterogeneous systems, based on our Matrix framework for resource management. It integrates local QoS mechanisms of the involved devices that deal mostly with short-term resource fluctuations, with a global adaptation mechanism that handles structural and long-term load variations on the system level. We have implemented the proposed approach and demonstrated its effectiveness in the context of video streaming.

Keywords: Quality-of-Service adaptation, distributed resource management, heterogenous systems, networked architectures, resource limitations and fluctuations

1 Introduction

In distributed heterogeneous environments, such as in-home entertainment networks and mobile computing systems, independently developed applications share common resources, e.g., CPU, network bandwidth or memory. The resource demands coming from different applications are usually highly fluctuating over time. For example, video processing results in both temporal fluctuations, caused by different coding techniques for video frames, and structural fluctuations, due to scene changes [1]. Similarly, wireless networks applications are exposed to long-term bandwidth variations caused by other application in the system that are using the same wireless network simultaneously, and short-term oscillations due to radio frequency interference, like microwave ovens or cordless phones. Still, applications in such open, dynamic and heterogeneous environments are expected to maintain required performance levels.

Quality-of-Service (QoS) adaptation is one of the crucial operation to maximize overall system quality as perceived by the user while still satisfying individual application demands. It involves monitoring and adjustment of resources and data flows in order to ensure delivering of certain performance quality level to the application. This can be done *locally* on a device, i.e., local resource adaptation mechanisms on devices detect changes in resource availability and react to them by adjusting local resource consumption on host devices, or *globally*, on the system level, i.e., the QoS adaptation is performed by a global QoS manager with a full knowledge of the system resources. The first approach has the advantage that the application can use domain specific knowledge to adapt its execution to the available resources. For example, in a video streaming application, this could be achieved by decreasing the stream bit rate or skipping video frames. On the other hand, a global resource management is aware of the demand of other applications and it has an overview of the total resource availability on the system level. In this way, it may reassign budgets, or negotiate new contracts to maximize system overall performance.

While most of the existing approaches provide mechanisms for either local or global adaptation, we believe that both methods should be used together in order to respond properly to both local and global fluctuations. Hence, we propose an *integrated* global and local QoS adaptation mechanism, where the structural and long-term load variations on the system level are object for global adaptation, while the temporal load and short-term resource variations are taken care locally on devices. The task of the local adaptation mechanism is to adjust resource usage locally on a device as long as the fluctuation is kept within a certain QoS range. If the resource usage exceeds the range's threshold, the global adaptation mechanism takes over and performs resource reallocation on the system level.

QoS-aware applications are usually structured in such a way that they can provide different discrete quality levels, which have associated estimations of the required resources. We use the notion of abstract quality levels for defining QoS ranges, such as *high*, *medium* and *low* resource availability. This provides a general QoS framework that is not application or device specific. As long as an application or a device can express its resource demand and consumption in terms of an abstract level, it can benefit from our method.

Implementation of the proposed integrated QoS mechanism is enabled by our previous work, the Matrix framework for efficient resource management in distributed, heterogeneous environments [2]. It provides a global abstraction of device states as representation of the system state for resource management and it decouples device scheduling and system resource allocation. In this work, we use the Matrix as the infrastructure to develop global and local adaptation mechanisms and to integrate them into a single QoS adaptation unit in a system. While some parts of the resource management mechanism are adopted from the original Matrix approach and further developed here in terms of the newly proposed global adaptation mechanism, the local adaptation and the integrated mechanism are entirely new contributions. Furthermore, the original parts of this paper include a new module in the Matrix, the application adapter used for application adaptation, the quality level mapping and interfacing, the closed-loop control model for resource monitoring and adaptation, as well as the deployment of our approach in the context of video streaming and video stream adaptation.

The rest of this paper is organized as follows. In the next section we give an overview of the related work. In Section 3 we describe the extended Matrix framework used in our approach. In Section 4 we describe the global and the local adaptation mechanism and show how to integrate them in a single approach. In the same section, we present an example of how our method can be used in the context of media streaming. In Section 5 we describe the current implementation status, followed by Section 6, which concludes the paper.

2 Related Work

Comprehensive work on distributed QoS management architectures has been presented in [3–7]. However, those architectures are mostly designed to work over networks like ATM or the Internet with Integrated Services (IntServ) and Differentiated Services (DiffServ) support, i.e., networks that can provide guarantees on bandwidth and delay for data transfer. In our work, we do not make any assumptions that the underlying system (OS or network) can offer any QoS guarantees. We consider a distributed, heterogeneous environment where applications share resources, such as CPU and network bandwidth. Applications can either execute on a single device, or on several devices (e.g., a video streaming application that involves reading a stream on a server and sending it through a network to a hand held device to be decoded and displayed). Furthermore, we assume that handovers and other network related issues are done by lower level in the system architecture, and those are not task of our research.

While architectures like [8] give an overall management system for end-to-end QoS, covering all aspects from a user QoS policies to network handovers, in our work we focus on QoS management and resource adaptation in application domain. Our work is related to

[9, 10], which present application-aware QoS adaptation. Both of them make a separation between the adaptations on the system and application levels. While in [9] the application adjustment is actively controlled by a middleware control framework, in [10] this process is left to the application itself, based on upcalls from the underlying system.

Our work differs in how an application adaptation is initiated and performed. We do adaptation on different architectural levels, but unlike the mentioned work, we address global and local adaptation and the integration of both approaches. We perform global adaptation of all resources within the system, while the work above focus on adjustment of resources on the end system, where the adaptation is based on the limited view of the sate of one device. We also provide an application independent approach, i.e., it can be used with different types of applications. Furthermore our approach can support component-based, decoupled approaches, where different components, like CPU or network schedulers can easily be replaced.

More recently, control theories have been examined for QoS adaptation. The work presented in [11] shows how an application can be controlled by a task control model. Method presented in [12] uses control theory to continuously adapt system behaviour to varying resources. However, a continuously adaptation maximizes the global quality of the system but it also causes large complexity of the optimization problem. Instead, we propose adaptive QoS provision based on a finite number of quality levels.

3 Resource Management Framework

Here we present the resource management framework used in our QoS adaptation method. First we give an overview of our previous work on distributed resource management, and then we extend it to suit the needs of the integrated QoS adaptation approach that will be presented in the next section.

3.1 Matrix framework

The Matrix is an adaptive framework for efficient management of resources in distributed, heterogeneous environments. Figure 1 shows the data flow (information flow) between the Matrix components. The *Resource Manager* (RM) is used to globally schedule and reserve resources in the system, i.e., it makes decisions for resource usage for all devices in the system. Likewise, each time a new application is about to enter the system, the RM performs admission control.

For example, in a video streaming application, if the display device, e.g., a PDA, cannot manage to decode and display all video frames on time, the Resource Manager will notice this and instruct the sender device to send a less demanding version of the stream (e.g., with lower resolution).

In order to deal with resource reservation, the Resource Manager has to have knowledge about currently available resources in the system. This is provided in the *Status Matrix* (SM). For the example above, the Status Matrix will contain the information that CPU availability on the PDA is low while the bandwidth for the wireless link between the streaming server and the PDA device is high. The SM also provides information about active applications resource requirements, priorities, sink and source destinations.

Based on the information stored in the Status Matrix, the Resource Manager will make decisions for resource reallocation in the system, and store the orders for devices the *Order Matrix* (OM). An example of such an order could be one given to the streaming server to decrease the quality of streamed video.

The resource status information in the Status Matrix is provided by the *Order Managers* (OMR), located on the devices. For each type of shared resources, there is an Order Manager responsible for publishing the current resource availability on the device in the Status Matrix. This information is provided to the Order Manager through *Local Monitors* (LM),

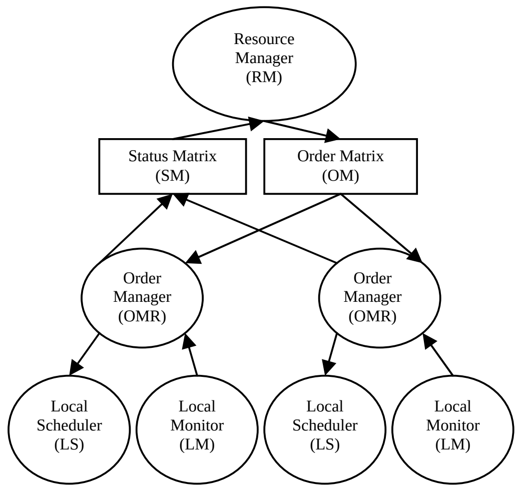

Fig. 1. The Matrix: Information flow

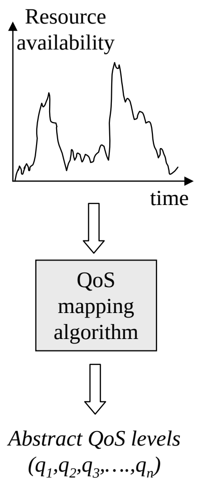

Fig. 2. Abstract QoS levels

that are responsible for continuous monitoring of a resource availability on a device, e.g., the available CPU or the network bandwidth. The accuracy of the information depends on a chosen temporal granularity.

Furthermore, an Order Manager receives orders from the Order Matrix and makes sure to adjusts local resource usage according to them. This is done through *Local Schedulers* (LS), which are responsible for scheduling of a local resources, e.g., a network packets scheduler that can adjust the packet sending rate according to available bandwidth.

For further details on Matrix framework we refer to our previous work [2, 13].

3.2 QoS levels

We want to use the minimum relevant information about devices states as needed for resource management, in order to reduce the system state presentation, and to abstract over fluctuations, which could overload scheduling of resources. Thus, we use the notion of a few *abstract QoS levels* that represent a resource's availability and an application's quality. For example, the variations in the quality of network link connection between two devices can be represented by e.g., three abstract QoS level values, (L)ow, (M)edium and (H)igh. H means that the data can be transmitted through the link with full available capacity, while L indicates severe bandwidth limitations. Likewise, quality of each application using certain resources is mapped to a finite number of application QoS levels.

In general, the availability of each resource is represented in our approach as a vector of discrete range of *n* QoS performance levels $\{q_1, q_2, ...q_k, q_{k+1}, ..., q_n\}$, see Figure 2. The value range of a QoS level q_k is defined by its threshold values [q_k^{min}, q_k^{max}].

In this work, we apply linear mapping between the resources and the QoS levels, e.g., based on experimental measurements [14]. For example, one simple mapping for the CPU bandwidth based on the CPU utilization U could be e.g., $0 \leq U \leq 0.3 \Rightarrow H$, $0.3 < U \leq 0.6 \Rightarrow M$, $0.6 < U \leq 1.0 \Rightarrow L$. A more advanced mapping could, for instance, use fuzzy logic to provide a larger number of QoS levels with finer granularity, but QoS mapping is an ongoing work and it is out of the scope of this paper.

3.3 Application Adapter

The Matrix is an application independent framework, and application adaptation is not the main focus of our work. However, in order to advance the usage of the Matrix along with various types of applications, we have extended the original Matrix architecture with an additional component, the *Application Adapter* (AA). The Application Adapter performs the mapping of QoS levels to the application specific parameters, and vice versa. For example, the AA for a video streaming application could map abstract quality levels, such as H, M and L, into real possible frame-per-second (fps) values for the stream, e.g., for a 30 fps MPEG-2 stream high quality could mean the fps-interval between 24 and 30 fps, medium quality is 16 to 23 fps and low quality could be defined as 10 to 15 fps.

Since this process is application specific, our ambition was to provide an interface for this component, and than is up to the application designer to implement it. If there is a way in an application to map its resource fluctuations into some abstract levels, then it can be used with our design. Also, upon resource reallocation, the Application Adapter will receive orders about new abstract levels from the Order Manager, which must be translated into some concrete actions on the application level.

4 Integrated QoS Adaptation Approach

In this section we present our integrated global and local adaptation mechanism that uses the Matrix framework. In our approach, global adaptation is performed by the Resource Manager, while the local adaptation is taken care of locally on the devices.

Consider the following motivating example: A person uses a PDA to watch a video stored on a local video server, which is delivered to the PDA through a wireless network. As the person moves around with the PDA, at some point it becomes almost out of range for the server, which results in video interruption due to packet losses. A local adaptation on the PDA does not really help in this case, since the video disruption is caused by the buffer underflow in PDAs decoder (in the case of buffer overflow, this could be treated locally on the PDA by e.g., speeding up the video decoding task). However, if there is a mechanism at the system level that can detect the lower bandwidth of the wireless link, i.e., the Matrix framework described in previous section, it could instruct the video server to stream a lower quality video stream that takes less network bandwidth.

Expressed in more general terms, resource consumption is adjusted locally on devices as long as the fluctuation stays within the range of requested QoS. For example, the Local Monitor detects a change in available CPU for a certain application, but this change is not large enough to enforce a different quality level to the application. Instead, the Local Scheduler could perform some local countermeasures, e.g., prioritize the application on the cost of some other application running on the same device. However, if the resource availability passes the defined thresholds (abstract QoS levels), the entire system gets involved via the global adaptation mechanism. The whole idea is illustrated in Figure 3.

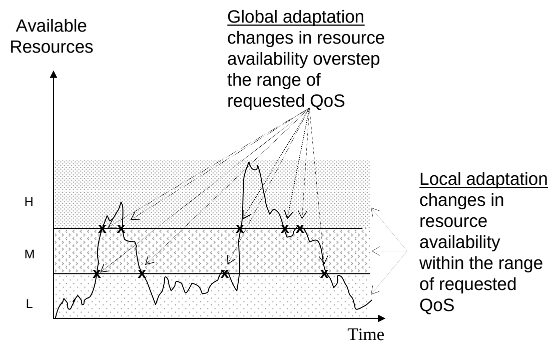
Available Resources
Global adaptation
changes in resource availability overstep the range of requested QoS
H
M
L
Local adaptation
changes in resource availability within the range of requested QoS
Time

Fig. 3. Different types of resource variations handled on different architectural levels

4.1 Local Adaptation Mechanism

Local adaptation involves detecting the changes in resource availability and reacting to those via the local scheduler. The ideas from control theory can be used to achieve this. We use the *closed loop* model, i.e., a control model that involves feedback to ensure that a set of conditions is met. It involves the Local Monitor, the Local Scheduler, and the Order Manager, see Figure 4. Expressed by terminology of the control theory, we use the following terms for inputs and outputs variables in our control model; *control variable*, v_{ctrl}, is the value observed by the local monitor (e.g. network packet loss, CPU utilization), *reference variable*, v_{ref}, is concrete performance specification for Local Schedulers made by the order manager, *error* ε is the difference between the value observed by the Local Monitor and the reference variable, and *control input variable*, v_{in}, is the value calculated by the adaptation algorithm in order to adapt scheduling of the local resources. The Local

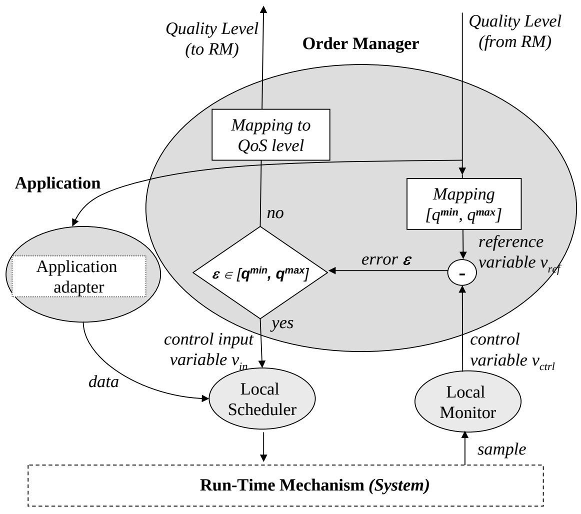
Quality Level (to RM)
Order Manager
Quality Level (from RM)
Mapping to QoS level
Application
Mapping [q^{min}, q^{max}]
no
reference variable v_{ref}
Application adapter
$\varepsilon \in [q^{min}, q^{max}]$
error ε
-
yes
control input variable v_{in}
control variable v_{ctrl}
data
Local Scheduler
Local Monitor
sample
Run-Time Mechanism (System)

Fig. 4. Local QoS Adaptation Mechanism

Monitor continuously monitors available resources in the system (e.g., CPU or bandwidth). Thus, in our control model it acts as an *observer* of the controlled system. It send the observed control value to the Order Manager. The Order Manager calculates the difference between the desired value, defined by the currently used QoS level, and the observed control value, i.e., it calculates the error value of the control loop. As long as resource availability stays within the boundaries for the given QoS level, i.e., the error falls in the range of the current QoS level, the output of the adaptation algorithm, control input, is passed to the Local Scheduler, i.e., the adapter part of control loop.

In the case that the error value implies a change in QoS levels, the values in the Status Matrix are updated and the Resource Manager is informed about the change. From this point, the global adaptation mechanism takes over, which we describe next.

4.2 Global Adaptation Mechanism

Whenever a local mechanism detects that a local resource availability has exceeded the current QoS level, a global adaptation mechanism will be initiated. The objective of the global adaptation is to adjust the resource usage among all involved applications. If the resource availability has increased, it will be given to involved applications (in terms of increased quality levels). Similarly, if the resource availability has decreased, the quality levels of the consumer applications will be decreased.

We support user defined *priorities* to be used when redistributing resources, i.e., the higher the priority of an application, the faster the quality increase of the application. However, it is up to the user to use priorities or not. Based on this, we distinguish between three reallocation policies in our approach, *fair*, *fair prioritized* and *greedy*.

Fair reallocation – If the priorities are not used, then the resources are adjusted (increased or decreased) in a strictly fair fashion: for each consumer applications the quality is adjusted step-by-step, one QoS level at the time, and then, if there are still resources to increase/decrease, we repeat the procedure for all applications once again, until the resource is consumed/replanished. For example, consider four different applications a_1,a_2, a_3 and a_4 that are using the same resource r. The current quality level for each applications is set to L. Assume that a_4 gets terminated and the resource availability of r gets increased by the portion used by a_4. The freed resource is given back to the remaining three application such that we first increase the the QoS level of a_1,a_2 and a_3 to M, and then, if there are still resources left, all QoS levels are increased to H.

Fair-prioritized reallocation – Note that in the fair approach, there is no guarantee that a certain application will change its QoS level. In the example above, there could be a case where the freed resource is entirely consumed after increasing the level of a_1 and a_2 to level M, so that a_3 will remain running on level L, despite the fact that a_3 might be the most important one in the system. However, if we use priorities, we could instruct RM to start by increasing the QoS levels of high priority applications first, i.e., a_3 in the example above. In other words, the resources are reallocated in a fair fashion, i.e., each application's quality level is changed by one step before changing any other application's level one more step, but also we use priorities to determine which applications should be served first.

Greedy reallocation – Moreover, priorities enable for an another reallocation policy, i.e., greedy redistribution. This means to increase (decrease) QoS level of an application with the highest (lowest) priority until it reaches its maximum (minimum) QoS level, before we start with the next one application (in the priority order). For the example above, we would continue increasing the QoS level of a_3 until it reaches H, before doing any QoS increase of a_1 and a_2. Furthermore, the priorities can be used when selecting which applications to drop first if that becomes necessary.

If an application is processed by several different devices, then, before changing its quality level, we need to check if the new level can be supported by all involved devices on the application's playout route. For example, in a video streaming application where a video stream is sent from a video server to a hand held device via a laptop, the bandwidth increase between the server and the laptop does not necessarily mean that we should start streaming a higher bit rate stream, since the link between the laptop and the hand held device might not be able to support it. Likewise, we have to consider if this increased quality can be supported by all other types of resources that the application is consuming e.g., there is no point to send more data over the communication link than it cannot be timely processed at the receiver device (by the local CPU).

Our *admission control* approach for new applications is quite similar to the adaptation approach described above. Thus, each time a new application is about to enter the system, the Resource Manager has to determine if sufficient resources are available to satisfy the desired QoS of the new connection, without violating QoS of existing applications. If yes, then we accept the new application and publish orders for resource reservation/reallocation into the Order Matrix. If no, we check if there are any existing application with the the lower priority than the new one, and if so, decrease their QoS (starting with the lowest priority application) to free some resources for the new application. If there are no available resources, and no lower priority applications, the new applications is rejected.

4.3 Pseudo-Code for Integrated Approach

Here is the pseudo-code for our current implementation of the integrated local and global QoS adaptation mechanism. We introduce some additional terms, as a complement to the terms presented earlier:

– $\mathcal{A} = \{a_1, a_2, .., a_n\}$, a set of applications in the system.
– $\mathcal{R} = \{r_1, r_2, ..., r_m\}$, a set of resources in the system.
– $\mathcal{D} = \{d_1, d_2, ..., d_p\}$, a set of devices in the system.
– $\mathcal{A}(r_i) \in \mathcal{A}$, a subset of applications that currently use resource r_i.
– $\mathcal{R}(a_j) \in \mathcal{R}$, a subset of resources currently used by application a_j.
– $\mathcal{R}(d_l) \in \mathcal{R}$, a subset of resources currently consumed on device d_l.
– $\mathcal{D}(a_j) \in \mathcal{D}$, a subset of devices currently used for processing of application a_j.
– $S(r_i)$, current resource supply (availability) of resource r_i.
– $D(r_i)$, current resource demand of all applications using r_i.
– $q_k(r_i)$ and $q_k(a_j)$, the *k*-th QoS level of resource r_i, respective application a_j, as described in section 3.2.

```
/* For the sake of simpler explanation, we omit in the pseudo-code for the start up activities where the devices has reported the local resource availability, and the RM has published initial QoS levels in the Status Matrix */

$\forall\, d_i \in \mathcal{D}$    /* For each device */
  $\forall\, r_i \in \mathcal{R}(d_i)$    /* For each resource on a device */

    /* Invoke local adaptation based on the currently assigned quality level */
    map $q_k(r_i) \Rightarrow [q_k^{min}(r_i), q_k^{max}(r_i)]$
    $v_{ref} = q_k^{max}(r_i)$, $\varepsilon^{max} = q_k^{max}(r_i) - q_k^{min}(r_i)$
    Do
      get $v_{ctrl}$ from LM
      $\varepsilon = v_{ref} - v_{ctrl}$
      calculate $v_{in}(\varepsilon)$ and send it to LS
    While $(0 \le \varepsilon \le \varepsilon^{max})$

    /* Prepare for global adapt. when the error exceeds the limit of current QoS level */
    map $\varepsilon \Rightarrow q_l(r_i)$, $l \neq k$
    publish $q_l(r_i)$ in SM
      $\Rightarrow$ break! invoke global adaptation

/* RM performs global adaptation based on new info in SM */

/* Case 1: total resource supply is greater than the total demand $\Rightarrow$ increase QoS levels */
If $(S(r_i) > D(r_i))$ Then
  Do
    /* Based on the chosen realloc. policy get an application to increase its QoS level */
    If ($a_j$ = getApplication(POLICY, INCREASE)) Then

      /* Check if all $a_j$'s proc. devices (other than $d_i$), support the next QoS level of $a_j$ */
      If($\forall d_j \in \mathcal{D}(a_j), d_j \neq d_i, d_j$ supports $q_{k+1}(a_j)$) Then

        /* Check if the new QoS level of $a_j$ can be served by all other $a_j$'s resources*/
        If($\forall r_n \in \mathcal{R}(a_j), r_n \neq r_i, r_n$ supports $q_{k+1}(a_j)$) Then
          increase quality of $a_j$ to $q_{k+1}(a_j)$
          /* incr/decr dem/sup for $r_i$ by the amount used to jump to next QoS lev.*/
```

```
            $\Delta = q_{k+1}^{max}(r_i) - q_k^{max}(r_i)$
            $D(r_i) += \Delta; \quad S(r_i) -= \Delta$

    While ($S(r_i) > D(r_i)$ AND $a_j \neq$ NULL)

/* Case 2: total resource supply is less than the total demand $\Rightarrow$ decrease QoS levels */
Else
    /* Similar as above, but the QoS levels are decreased. Also, we do not need to check
    other devices and resources, since the decr. quality will not put extra demands on them.
    ...(omitted)
```

4.4 Example

Here we illustrate our approach in the context of video streaming. Consider the example scenario with the PDA and the streaming video server from Section 3, where the quality of the streamed video was dependent on the distance between the PDA and the server. At some point in time, the PDA is so far away from the server so it only makes sense to stream a low quality video stream, i.e., stream S_1 with the abstract quality level L and priority p_1. Assume also that there is another video stream in the system, S_2, streamed from the server to a laptop with a quality level H and higher priority p_2. The CPU availability (bandwidth) on all devices is initially assumed to be high. The reallocation policy used is fair-prioritized. The whole situation is depicted in Figure 5. The values within the parentheses are the new QoS levels (obtained after adaptation).

Now, assume that the person with the PDA starts moving closer to the server. The local adaptation mechanism on the one of the involved devices, i.e., either on the server or on the PDA, will detect that more and more packets can be sent between them (let's assume the PDA will detect this first). As the PDA is coming closer to the server, at some point the quality of the link connection will exceed the assigned threshold for the local adaptation, and the global adaptation mechanism will take over, with the following steps involved (see Figure 5 in parallel; the numbers below correspond to the numbers in the figure; some of the steps are merged):

1. The Local Monitor on the PDA detects that the link quality between the server and the PDA has increased.
2. This is reported to the Order Manager, who will map the new values to the quality level H (we can assume a sudden large connection improvement e.g., by entering the room where the server is placed).
3. Order Manager publishes the new quality level H in the Status Matrix.
4. Assume also that there has been some change in the CPU availability on the laptop, i.e., it gets decreased from H to L due some new, CPU intensive application that has started to run on the laptop. Initially, the local adaptation mechanism on the laptop will react to the changes in the CPU load by e.g., by performing selective frame skipping in the video decoder that is processing the stream S_2. However, at some point the CPU QoS threshold will be exceeded and the new QoS value will be calculated and published in the Status Matrix for the CPU.
5. The Resource Manager is notified about the new quality level values.
6. Now, it is up to the Resource Manager to take a decision about the resource reallocation. Considering the available bandwidth and the streams priorities, one solution could be to set the quality of S_1 to M (since it has lower priority), and left the quality of S_2 unchanged. However, streaming the high quality video stream to the laptop may not be a good solution, since the CPU on the laptop is overloaded and video frames will be skipped anyway. Hence, the Resource Manager, who has the total resource usage

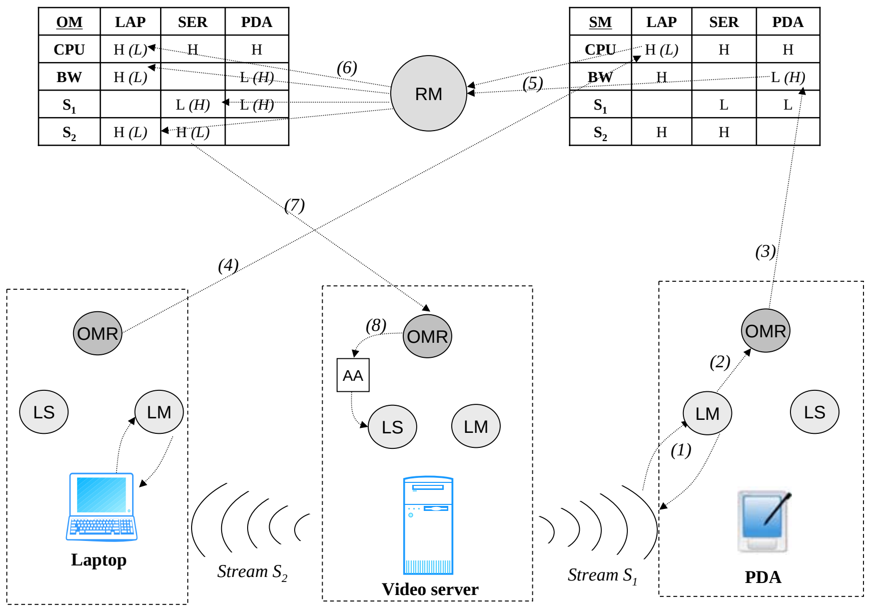

Fig. 5. Example global adaptation

view of the system, decides to set L for stream S_2. This decision will not only reflect the resource status on the laptop correctly, but also it will allow for S_1 to be set to H (which can be done because the quality of the connection between the server and the PDA has been changed to H).

7. The Order Managers on respective devices are informed about the new values (arrows to the OMRs of the PDA and the laptop are omitted in the figure to ease readability).
8. The Order Managers then enforce the new settings via their local schedulers and application adapters. For example, in the case of the server, the stream application adapter will make sure to decrease the quality of stream S_2. This can be done in several ways, e.g., by reading a lower quality version of S_2 that has been stored on the server in advance, or by using an online modification of original S_2 by using the quality-aware preventive frame skipping methods that we have developed in our previous work [15].

5 Implementation and Evaluation

The Matrix framework is quite complex and we are still working on its full implementation. However, we have implemented a mock-up of Matrix approach [2] using HLA [16]. Moreover, some basic benefits of our method, has been demonstrated by simulations.

5.1 Implemented modules

The hierarchical architecture and the loose coupling between system modules makes it possible to work on different parts independently. Current implementation includes Local Monitors and Schedulers for CPU and network bandwidth, and a Video Stream Adapter.

Local Network Scheduler – For network scheduling we use the traffic shaping approach, which provides different QoS by dynamically adapting the transmission rate of nodes, to match the currently available bandwidth of a wireless network. The Traffic Shaper adjusts the outbound traffic accordingly to input parameters (i.e., the amount of available bandwidth assign to the Local Scheduler). Please see [14] for full implementation details.

Local Network Monitor – For monitoring and estimation of available bandwidth (over 802.11b wireless Ethernet), we use a method that provides us with the average bandwidth that will be available during a certain time interval. The architecture consists of a bandwidth predictor that first uses a simple probe-packet technique to predict the available bandwidth. Then, exponential averaging is used to predict the future available bandwidth based on the current measurement and the history of previous predictions, see [14] for details.

Local CPU Scheduler – The allocation of CPU to the applications depends on the scheduling mechanism that is used. We have developed a predictable and flexible real-time scheduling method that we refer to as *slot shifting* [17]. The basic idea is to guarantee a certain quality of service to applications before run-time, and then adjust it at run-time according to the current status of the system.

Local CPU Monitor – Since we use a real-time scheduling mechanism, the CPU monitoring is very simple to achieve. The *spare capacity* mechanism of slot shifting provides easy access of the amount and the distribution of available resources at run-time [17].

Video Stream Adapter – We have implemented an Application Adapter for MPEG-2 video stream adaptation, based on quality-aware, selective frame skipping. Order Manager sends allowed abstract quality level to the video adapter, which then adjusts the stream according to available resources by skipping the least important video frames. For the frame priority assignment algorithm we have proposed a number of criteria to be applied when setting priorities to the frames. Please see our previous work [15] for details.

5.2 Evaluation

We have evaluated our method in the context of video streaming. Here we present results from a 15 minutes video streaming simulation using our integrated approach for global and local adaptation. We simulate usage of 30 devices in the system and show how a MPEG-2 video stream is adapted based on current resource availability (network bandwidth). We use the following quality levels for available bandwidth (given in Mbps): $q_1(BW) = [q_1^{max}, q_1^{min}] = [1.5, 2.5]\ (L)$, $q_2(BW) = [q_2^{max}, q_2^{min}] = [2.5, 4]\ (M)$, $q_3(BW) = [q_3^{max}, q_3^{min}] = [4, 11]\ (H)$. Figure 6 shows that the local adaptation mechanism is deployed most of the time (77%), while the global mechanism is triggered only when necessary (23%), i.e., the QoS has changed that much that the system reallocation must take place.

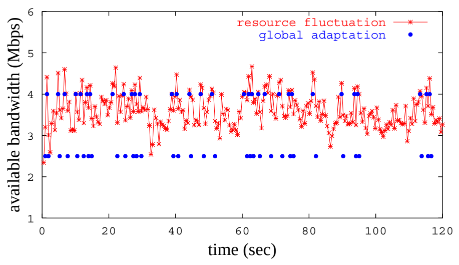

Fig. 6. Invocation of global adaptation

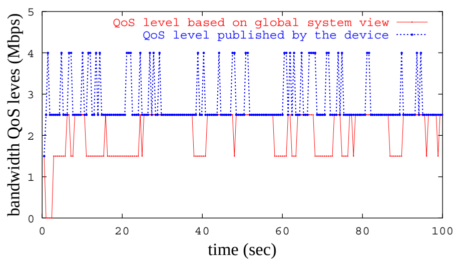

Fig. 7. Global vs Local system view

Figure 7 shows the difference between QoS levels based on one device's local view and those assigned by global adaptation, i.e. the possible spared resources (available bandwidth) on just one device due to global adaptation. It illustrates efficiency of our integrated approach where adjustment of resources is not just based on the limited local system view of one device, but also on the current available resources of all involved devices. In that way, our approach enables a system wide optimization.

6 Conclusions and Future Work

We proposed a method for efficient Quality-of-Service adaptation in dynamic, heterogenous environments. It integrates global and local adaptation, where the first one takes care of the structural resource fluctuations on the system level, while the second one is performed locally on devices to handle short-term variations. The idea is to perform local adaptation as long as possible, using a control model for resource monitoring and adjustment, and if a resource availability passes the range of the currently assigned QoS level, the global adaptation mechanism takes over.

Our current and future work include further developing the local control model by formally describing the system's behaviour with a set of differential equations. Futhermore, we are working on a more general model for mapping between resources demands and abstract QoS levels and exploiting the proposed framework in other application domains than in-home networks.

References

1. Otero Perez, C., Steffens, L., van der Stok, P., van Loo, S., Alonso, A., Ruz, J.F., Bril, R.J., Garca Valls, M.: QoS-based resource management for ambient intelligence, Ambient intelligence: impact on embedded system design, Academic Publishers Norwell, MA, USA, (2003)
2. Rizvanovic, L., Fohler, G.: The MATRIX: A QoS Framework forStreaming in Heterogeneous Systems, International Workshop on Real-Time for Multimedia, Catania, Italy, (2004)
3. Nahrstedt, K., Smith, J.M.: Design, Implementation an Experiences of the OMEGA End-Point Architecture, Distributed Systems Laboratory, University of Pennsylvania, Philadelphia
4. Nahrstedt, K., Chu, H., Narayan, S.: QoS-Aware Resource Management for Distributed Multimedia Applications, UIUCDCS-R-97-2030, (1997)
5. Campbell, A., Coulson, G., Hutchison, D.: A quality of service architecture, ACM SIGCOMM Computer Communication Review, (1994)
6. Gopalakrishna, G., Parulkar, G.: Efficient Quality of Service in Multimedia Computer Operating Systems, Washington University, (1994)
7. Shankar, M., De Miguel, M., Liu, J.W.S.: An end-to-end QoS management architecture, Real-Time Technology and Applications Symposium, (1999)
8. Kassler, A., Schorr, A., Niedermeier, C., Schmid, R., Schrader, A.: MASA - A scalable QoS Framework, Proceedings of Internet and Multimedia Systems and Applications (IMSA), Honolulu, USA, (2003)
9. Li, B., Nahrstedt, K.: A Control-Based Middleware Framework for Quality-of-Service Adaptations, Selected Areas in Communications, IEEE Journal, (1999)
10. Noble, B.D., Satyanarayanan, M., Narayanan, D., Tilton, J.E., Flinn, J., Walker, K.R.: Agile Application-Aware Adaptation for Mobility, 16th ACM Symposium on Operating Systems Principles, France, (1997)
11. Li, B., Nahrstedt, K.: Impact of Control Theory on QoS Adaptation in Distributed Middleware Systems, American Control Conference, (2001)
12. Stankovic, J.A., Abdelzaher, T., Marleya, M., Tao, G., Son, S.: Feedback control scheduling in distributed real-time systems, RTSS, (2001)
13. Rizvanovic, L., Fohler, G.: The MATRIX - A Framework for Real-time Resource Management for Video Streaming in Networks of Heterogenous Devices, Conference on Consumer Electronics, Las Vegas, USA, (2007)
14. Lennvall, T., Fohler, G.: Providing Adaptive QoS in Wireless Networks by Traffic Shaping, Resource management for media processing in networked embedded systems (RM4NES), Netherlands, (2005)
15. Isovic, D., Fohler, G.: Quality aware MPEG-2 Stream Adaptation in Resource Constrained Systems, ECRTS, Catania, Italy, (2004)
16. IEEE Standard for Modeling and Simulation, High Level Architecture (HLA) - Federate Interface Specification, No.:1516.1-2000
17. Isovic, D., Fohler, G.: Efficient Scheduling of Sporadic, Aperiodic, and Periodic Tasks with Complex Constraints, 21st IEEE RTSS, USA, (2000)