Handling overload of target sensitive real-time applications for increased system utility and improved resource usage

Raphael Guerra and Gerhard Fohler
Chair for Real Time Systems
Technische Universität Kaiserslautern, Germany
[guerra,fohler]@eit.uni-kl.de

ABSTRACT
The gravitational task model allows target sensitive applications (e.g. multimedia and control) to express target points for maximum utility, and a utility decay as a function of the deviation from these points. The compromise among their deviations for maximized utility is calculated based on a physical pendulum analogy. However, in an overloaded system a compromise among all jobs does not necessarily incur in maximized utility.

In this paper we propose an overload handling mechanism for the gravitational task model which accounts for both system load and target sensitivity of applications. This mechanism considers the trade-off between aborting and shifting the execution of a job for resulting increased utility accrual and improved resource usage. The heuristic to abort jobs is based on their utility and amount of consumed resources. An aborted job does not yield any utility, but frees resources that can be exploited by other jobs for increased utility accrual.

A multimedia case study shows that our mechanism reduces frame display jitter and improves resource usage.

Categories and Subject Descriptors
3. [Real-Time Systems]: Miscellaneous

General Terms
Scheduling

Keywords
adaptivity, target sensitivity, overload, utility

1. INTRODUCTION
Real-time tasks must provide functionally correct results and meet timing constraints for correct system behavior; these timing constraints vary among applications. Target sensitive real-time applications must execute within a time interval bounded by earliest start time and deadline constraints to yield some utility to the system, and have this utility maximized when executing at a target point. The system is overloaded if at least one job cannot execute at its target point. In this case, a compromise for the deviation of each job from its target point must account for maximum system utility accrual.

There is no direct relation between the amount of jobs that execute and the final accrued utility. For example, jobs that do not meet the start-time-deadline constraints do not accrue any utility to the system. The abortion of a job frees resources which other jobs may use to decrease their deviation from their target point, hence yielding an extra amount of utility. The trade-off between abortion and deviation from the target point must account for a resulting increased utility accrual, which is a NP-hard problem. Notice that abortion/deadline-miss might also contribute negatively to the system utility, e.g. in the case of hard real time tasks.

Examples of target sensitive real-time applications include multimedia and control [7]. In media processing, frames must be displayed at target points for the best user perceived quality of the video stream. Buffering frames in advance is not an option in these applications [12], hence demanding frames to be displayed before the next frame starts being decoded. If frames cannot be displayed at their target points, e.g., due to the execution of other jobs, they can be either dropped or delayed. The impact of dropping/delaying the display of a frame on the perceived quality of video may vary among frames [12]. In control, sampling and actuation are ideally executed at their target points for optimum output [13]. Shifting them a little bit for the sake of feasibility is acceptable provided the system response remains within bounded limits. On the other hand, jitter in the computations executed between samplings and actuations do not jeopardize the system.

In the ACTORS framework [9], both multimedia and control applications run concurrently on mobile multimedia devices with limited resource availability; the resource manager [14] of this framework uses control routines to monitor resource utilization and multimedia applications are initiated by the user. Overload happens often in such systems and the accrued utility directly relates to the perceived quality of video by the user and accuracy of the resource utilization monitoring.

There are many approaches in the literature to handle
overload in real-time systems, but most of them focus on meeting deadline constraints [3, 10, 5, 4, 2]. The elastic task model [5], for instance, handles overload conditions by changing the periods of tasks to adjust the system utilization so that the task set remains feasible. Those approaches implicitly assume all tasks accrue the same utility to the system, hence drawing a direct relation between the amount of jobs that execute and the final accrued utility. The Generic Utility Scheduler (GUS) [1] and the utility-based packet scheduling [15] provide a time-utility based overload handling mechanism based on the potential utility density of jobs, but do not consider target sensitivity.

The gravitational task model [6, 7, 8] allows tasks to express a target point for execution and a utility decay as a function of the deviation from this point. Work done on this model includes: finding a compromise among the deviation of all jobs for increased utility accrual which is based on the equilibrium of physical pendulums; a heuristic to order jobs’ execution; and, an on-line scheduling algorithm which uses the equilibrium. However, as of now no work considers the trade-off between abortion and the deviation from the target point.

In this paper we propose an overload handling mechanism for target sensitive real time applications. This mechanism differs from previous work because here we consider the trade-off between aborting and shifting the execution of jobs in order to account for increased system utility. The abortion heuristic of our mechanism is based on the utility density of jobs, which is the ratio between utility accrual and execution time.

Our mechanism has linear complexity and is application independent; any application can benefit from it by modeling its requirements into the parameters of the gravitational task model. The evaluation section brings a multimedia case study where this mechanism is used to reduce frame display jitter and improved resource usage.

The rest of this paper is structured as follows: in section 2 we recall the gravitational task model as needed for the rest of this paper; in section 3 we describe our overload handling mechanism and demonstrate it with an example; section 4 presents our experimental results from a multimedia case study; finally, section 5 draws the conclusions and future work.

2. THE GRAVITATIONAL TASK MODEL

In this section we briefly recall the gravitational task model [6, 8, 7], which expresses the compromise between conflicting timing constraints of different jobs based on the equilibrium state of physical pendulums. A pendulum is an object attached to a pivot point that can swing freely. A basic example is the simple gravity pendulum or bob pendulum. As depicted in Figure 1, it consists of a bob at the end of a massless string, which, when given an initial push, will swing back and forth under the influence of gravity over its central (lowest) point in a circular trajectory. Placed in the lowest point, the bob will come to rest there (rest position). If the bob pendulum contains more than one bob, they cannot be all at the same time in the lowest point, and hence, will push each other aside to find a new rest position, i.e. the equilibrium state. The equilibrium condition is that the sum of all torques in the system is equal to zero and the distance between the centers of two consecutive bobs is the sum of their radii.

Figure 1: Analogy between bob pendulum and real-time tasks

The gravitational task model assumes jobs $j_i$ with earliest start time $est_i$, relative deadline $dl_i$, worst case execution time $WCET_i$, target point $tp_i$ and importance $imp_i$. The execution window of a job $j_i$ is defined as $[est_i, est_i + dl_i]$. A job obtains its highest utility at the target point; executing somewhat before or after is feasible, but at lower utility. Jobs are not allowed to execute outside their execution window. Each job can express an utility decay as a function of its deviation from the target point. Jobs may or may not be instances of recurring tasks. Finally, importance represents the flexibility of a job to be shifted from its target point in the presence of other jobs, i.e., the importance is proportional to the need of the job to execute at its target point.

Figure 2: Analogy between particle pendulum and real-time tasks

Drawing the analogy, we can think of a bob as a job whose execution time is equivalent to the size of the bob. A job is allowed to execute at its target point in the absence of other jobs in the system with the same target point. The target point is equivalent to the central lowest point of a pendulum trajectory and the swinging range is the execution window of the job. The importance of a job, which represents the resistance to shift around the target point, can be seen as the weight of the bob. The heavier a bob is, the closer to the bottom it will come to rest. Finally, the job utility as a
function of its point of execution is similar to the potential energy of a bob as a function of its position in its trajectory, i.e. an elliptical function. Table 1 summarizes this analogy.

<table>
<thead>
<tr>
<th>pendulum</th>
<th>task set</th>
</tr>
</thead>
<tbody>
<tr>
<td>bob</td>
<td>job</td>
</tr>
<tr>
<td>weight</td>
<td>importance</td>
</tr>
<tr>
<td>swinging range</td>
<td>execution window</td>
</tr>
<tr>
<td>central point</td>
<td>target point</td>
</tr>
<tr>
<td>potential energy function</td>
<td>utility function</td>
</tr>
<tr>
<td>equilibrium state</td>
<td>best compromise</td>
</tr>
</tbody>
</table>

Table 1: Analogy between bob pendulum and task set

Although the analogy described above appears straightforward, a number of issues have to be addressed before a direct mapping to the real-time task scheduling. See [7] for the detailed discussion and motivation. In the analogy depicted in figure 2, we consider a job as a particle (a massive point) instead of a rigid body (previously a bob). As the whole execution of a job cannot happen in one point in time, in this analogy a particle represents the anchor point of a job. The value of an anchor point ($\alpha_i$) is the portion of execution of job $j_i$ that executes before this anchor point ($0 \leq \alpha_i \leq 1$). Each particle must be at a constant horizontal distance $d_i$ from the next particle, where $d_i$ is the amount of execution time in between $\alpha_i$ and $\alpha_{i+1}$. The rest of the analogy remains unchanged. Each particle has a weight $W_i$ and hangs on a pivot point $P_i$ by a massless string of length $R_i$. All pivot points are aligned perpendicularly to the gravity.

The deviation of the last particle from its target point ($dev_N$) under the new distance constraint is solved with linear complexity using equation 1. By converting the task set variables into the particle pendulum environment as shown in table 2, this equation is used to approximate the compromise that maximizes the system’s accrued utility for a group of jobs in a busy period. Due to the lack of space, we advise the reader to refer to [7, 8] for a complete description of the equilibrium calculation and how the mapping shown in table 2 is done.

$$dev_N = \sum_{i=1}^{N-1} W_i \times (\sum_{j=1}^{N-1} d_{i,j}) + P_i - P_N$$

(1)

Table 2: Mapping task parameters into particle pendulum parameters

<table>
<thead>
<tr>
<th>pendulum</th>
<th>task set</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_i$</td>
<td>$2 \times imp_i / (d_i - WCET_i)$</td>
</tr>
<tr>
<td>$R_i$</td>
<td>$est_i + (d_{i+1} - WCET_{i+1})/2 + \alpha_i WCET_i$</td>
</tr>
<tr>
<td>$P_i$</td>
<td>$(1 - \alpha_i)WCET_i + \alpha_{i+1} WCET_{i+1}$</td>
</tr>
</tbody>
</table>

Table 2: Mapping task parameters into particle pendulum parameters

3. HANDLING OVERLOAD

The system utility is not only related to the amount of work done. If system resources are not enough to allow all jobs to execute at their target points (overload condition), a compromise for maximized utility accrual might imply the abortion of some jobs. Resources freed by an aborted job may allow other jobs in the system to accrue an extra amount of utility which results in an increase in the resulting system utility. Notice that an aborted job can either accrue no utility or accrue a negative utility to the system. In this work we will only consider the former case.

Our overload handling mechanism uses a heuristic which discards jobs based on their utility density and accounts for target sensitivity based on the equilibrium of the gravitational task model. The utility density is defined as the ratio between the maximum utility a job may accrue (i.e. its importance) and worst case execution time of a job. As in the gravitational task model all jobs have elliptical utility functions, the importance alone implicitly accounts for the utility variation as a function of the deviation from the target point. The smaller this ratio is, the smaller is the utility accrual of the job per unit of execution. If these units of execution are used by other jobs with higher utility density that compete for the same units of execution the resulting utility accrual might be higher (example in the end of this section).

The overload handling mechanism works as follows. Assume any existing scheduling algorithm based on the gravitational task model. Those algorithms constitute of an ordering phase, in which the order jobs execute is defined, and a timing phase, in which the execution of jobs is shifted using the equilibrium equation [7, 8]. Our overload handling mechanism maintains a job list with the order jobs execute ($exec_{list}$), a job list ordered by decreasing utility density ($density_{list}$), and a job list with the final schedule ($sched_{list}$). The sorting criteria for $exec_{list}$ depends on the scheduling algorithm. An incoming job is inserted in $exec_{list}$ and $density_{list}$ accordingly. Then, our mechanism inserts jobs in $sched_{list}$ in decreasing order of utility density at the position defined by $exec_{list}$, and applies the timing phase. Whenever inserting a job results in a smaller system utility, this job is aborted and the timing phase is reapplied. This process is repeated until there is no job left to be inserted in the schedule. See algorithm 1.

Algorithm 1 Overload handling mechanism.

```python
input: exec_list, density_list, incoming_job

sched_list = new_empty_list()
insert_sorted(incoming_job, exec_list)
insert_sorted(incoming_job, density_list)
for (i=0; i<\text{length}(density_list); i++)
{
    job = density_list[i]
    utility_before = utility(sched_list)
    insert_sorted(job, sched_list, exec_list)
    timing_phase(sched_list)
    if (utility(sched_list)<utility_before)
    {
        abort(job, sched_list)
        timing_phase(sched_list)
    }
}

return sched_list
```
The complexity to insert the incoming job in both lists is linear. The work in [8] shows a timing phase algorithm with linear complexity, given that the execution order is known. Therefore, the final complexity of this overload handling mechanism, including the scheduling algorithm, is linear.

Consider the following example, where we schedule the jobs described in table 3; in this example, for the sake of simplicity, we order exec_list by target point [6]. The execution order in the schedule is, then, j1, j2 and j3. Our overload handling mechanism inserts those jobs in the schedule in decreasing order of utility density. Therefore, we initially insert job j1 in the schedule directly at its target point. The accrued utility of the system is 10 at this point and this state is depicted in figure 3(a). Next, we insert j3 in the schedule, which can also be scheduled directly at its target point. The accrued utility of the system is 18 at this point and this state is depicted in figure 3(b). At last, we insert job j2 in the schedule, which demands a recomputation of the equilibrium of the system. With these job parameters, the equilibrium schedules job j1 at time 2.57, job j2 at time 3.57 and job j3 at time 5.57. This state is depicted in figure 3(c). In this state the accrued utility of the system is 17.5, which is smaller than the system utility without job j2. Therefore, the overload handling mechanism aborts the execution of j2, and the final schedule is as depicted in figure 3(b).

<table>
<thead>
<tr>
<th></th>
<th>j1</th>
<th>j2</th>
<th>j3</th>
</tr>
</thead>
<tbody>
<tr>
<td>start time</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>deadline</td>
<td>4</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>WCET</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>target point</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>anchor point</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>importance</td>
<td>10</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>util. density</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3: Job set.

Figure 3: The overload handling mechanism

Assume now that the importance of j2 is 6, hence having a utility density of 6. Applying the overload handling mechanism results in j1 executing at time 2.6, job j2 at time 3.6 and job j3 at time 5.6. In this schedule the accrued utility of the system is 21.5, which is more than the utility accrued when only j1 and j2 are scheduled to execute. Therefore, j3 (the job with lowest utility density) is not aborted.

4. EVALUATION

To the best of our knowledge, there is no previous work in the literature which accounts for the trade-off between aborting and shifting the execution of jobs for increased utility accrual. We believe that a comparison with approaches that consider exclusively either abortion or shifting is unfair and does not bring much value to the evaluation. Therefore, we evaluate our overload handling mechanism with a multimedia case study, which is an application example that can benefit from this trade-off. We first describe the case study, and then, show the simulation results.

4.1 Multimedia case study

High quality media processing, such as for consumer electronics, is target sensitive because each frame must be displayed at a specific point in time for maximized perceived quality of video (PQV). Common strategies to circumvent lack of resources are delayed frame decoding (and hence display) and frame skipping [12]. Here we show how our overload handling mechanism can express the trade-off between delaying and skipping the display of a frame for improved PQV and resource usage. For this case study, we consider the MPEG-2 and the MPEG-4 Simple Profile (SP) as the video compression standards. Notice that the capabilities of our overload handling mechanism are not limited neither to any particular standard nor to multimedia applications.

MPEG-2 is the standard used in DVDs and MPEG-4 SP is the most common high definition (HD) compression standard for Internet streaming and mobile devices nowadays. Among several other contributions, the MPEG-2 defines 3 different types of frames: I, P and B; MPEG-4 SP contains only I and P frames. While I frames are self-contained, P and B frames depend on other frames to be decoded. P frames depend on the previous P or I frame (which ever comes first) and B frames depend on the previous and the next P or I frame. This dependency graph is shown in figure 4(a). This strategy allows better compression by avoiding the replication of data that remains unchanged in between scenes, but imposes more challenges to the real-time aspect of the application. All frames in between two I frames (including the first I frame) comprise a GOP (Group of Pictures). A GOP finishing with a B frame is called dependent, else it is called independent.

Figure 4: Frame decoding and display

As discussed in section 2, future jobs that affect the schedule of jobs currently in the system must be considered in the equilibrium computation. For this case study we consider all frames belonging to the current GOP; we assume that all GOPs are independent. The start time of frames is the start of its GOP and the relative deadline is the length of the GOP. Having a large execution window gives the multimedia application more flexibility for adaptations under overload, yet allowing for maximum PQV under no overload. The anchor point of each frame decoding is 1, since we are interested in when the decoding is finished and the frame is ready for display. We assume the frame decoding time is known, e.g. given by a decoding time estimation tool.

Assuming a constant framerate, the target point of I and B frames is the start of the GOP plus the position of the frame in display order (see figure 4(a)) multiplied by the frame inter-display time. For example, suppose a frame rate of 25fps, which has an inter-display time of 1/25s = 40ms.
If the current GOP starts at time 1200\(\text{ms}\), its 3\text{rd} frame must be displayed at 1200 + 40 \times 3 = 1320\(\text{ms}\). For P frames, its position in the equation is subtracted by the number of B frames until the previous P or I frame in the display order. For example, the 4\text{th} frame of the GOP in figure 4(a), which is a P frame, has target point 1200 + 40 + (3 - 2) = 1240\(\text{ms}\). This way, ordering the execution of the decoding jobs by target point resolves all the dependencies. See figure 4(b).

Lastly, importances are assigned to the decoding jobs based on the frame skipping algorithm presented in [11]. As criteria we consider the frame type and the frame position in the GOP. The I frame is the most important in a GOP because all other frames are dependent on it. Next come the P frames, whose importances are dependent on their position in the GOP. The closer to the I frame in the same GOP, the higher is the importance. Last comes the B frames, whose importance are also based on their position on the GOP. However, since no frame depends on a B frame to be decoded, the importance here depends only on the frame distribution on the GOP. Skipping too many frames in a row affects the smoothness of the video, hence being preferable to give different importances to odd and even B frames; we assign higher priority to even frames.

In our importance assignment policy, even B frames have double the importance of odd B frames. Each P frame has double the importance of the next frame and the last P frame in the GOP has importance equal to the double of double the importance of the next frame and the last P frame in the GOP. Finally, the importance of the I frame is the double of the importance of the next P frame.

### 4.2 Experiments

![Jitter Dispersion](image)

**Figure 5: Jitter dispersion**

In our experiments we measure the percentage of skipped frames and the dispersion of the deviation from the target point. These parameters were chosen over a perceived quality of video analysis because, to the best of our knowledge, there is no existing method that can properly quantify it as a function of jitter and frame skipping. We evaluate the quality of our video adaptation strategy using 4 streams of different types, each one having between 100 and 200 GOPs and different properties. The 1\text{st} stream is the BBC news recorded from a Dreambox and has all GOPs with 12 frames. The 2\text{nd} stream is a tennis match recorded from EuroSport also using a Dreambox and with 12 frames in GOPs. The 3\text{rd} stream is a high definition documentary downloaded from the internet and has variable GOP sizes. Finally, the 4\text{th} stream is a scene of the movie Matrix recorded from a DVD, has no B frames, and has varying GOP sizes. For each stream, we assume 6 scenarios for the average CPU demand needed to decode all frames: 60%, 75%, 90%, 100%, 150%, 300%. The different scenarios are created by linearly scaling the original decoding times obtained from the trace files.

In figure 5 we show the boxplot with the deviation of each frame from the target point. In a boxplot, 50% of all points are located in the range of the box, being 25% above the thick line that crosses the box and 25% below. The whiskers above and below the box comprise, each one, 25% of all the points. As we can see in this figure, for all streams considered, if the average utilization is 60% or 75%, many of the frames finish their decoding exactly by their target point. This is concluded from the fact that the box comprising 50% of all jitter is very short and indicates a jitter of zero. Nonetheless, conflicting target points happen anyway and our method schedules the frames to keep the deviation under limited boundaries. As the CPU demand increases, the deviation increases in order to avoid skipping the whole movie. Even with the streams being so different, our method shows to be very stable based on the similarities of the 4 graphics.

![Fraction of Skipped Frames](image)

**Figure 6: Fraction of skipped frames**

In figure 6 we show the fraction of skipped frames in each of the streams for the CPU demand scenarios proposed. Since we consider average CPU demand, it is possible that some GOPs demand more CPU than others and for these GOPs, frames must be skipped even if resources are enough to decoded all frames by the end of the stream. Therefore, even in the case of CPU demand less than 100% we can observe some skipped frames. Once again, observing all streams analyzed, we can observe a stable behavior of our
method. Even for an average CPU demand of 300%, we observed that on average two frames are displayed per GOP, hence still allowing the user to appreciate a moving picture with some smoothness.

5. CONCLUSION

In this paper we presented a mechanism to handle overload of target sensitive applications for increased utility accrual and resource usage. This mechanism is based on the gravitational task model, which allows jobs to express a target point in time for maximized utility and a utility decay as a function of the deviation from this point. Previous work on the gravitational task model considered the order jobs execute and the compromise among the deviation jobs from their target points. Our approach here further considered aborting the execution of jobs; an aborted job does not yield any utility, but frees resources that can be exploited by other jobs for final increased utility accrual. We solved this NP-hard problem with a heuristic based on the utility of jobs and the amount of consumed resources.

We evaluated our approach with a video adaptation application for reduced frame display jitter and improved resource usage.

6. ACKNOWLEDGMENTS

The authors wish to express their gratitude to the members of the real-time systems group at TU Kaiserslautern for the good discussions and reviewing of the paper, notably Anand Kotra and Stefan Schorr. The authors acknowledge furthermore the comments of the reviewers of this paper.

7. REFERENCES