Toward a Realization of the Value of Benefit in Real-Time Systems

Lonnie R. Welch
Electrical Eng. & Computer Science
Ohio University
Athens, Ohio 45701
welch@ohio.edu

Scott Brandt
Computer Science Department
Jack Baskin School of Engineering
University of California at Santa Cruz
sbrandt@cse.ucsc.edu

Abstract

Real-time computing models that are based on benefit (also called utility and value) offer a generic paradigm that captures the spectrum from hard- to firm- to soft-real-time requirements. Furthermore, it allows robust, flexible real-time systems to be developed. Thus, it is the authors' opinion that benefit will become increasingly important in the theory and the practice of real-time computing. This paper discusses the notion of benefit in real-time systems and considers issues that must be addressed in order to fully exploit benefit-based models. It discusses how benefit is used in a variety of real-time paradigms and in example applications. It also identifies various types of benefit and presents a taxonomy that organizes the types.

1 Introduction

Researchers are broadening beyond the classifications of "hard-", "firm-" and "soft-" real-time, in favor of the general notions of benefit (also referred to as value and as utility). This growth is natural, as it allows general theories of real-time computing to be developed. However, this new frontier offers many challenges, including defining benefit and categorizing different types of benefit, modeling system requirements with benefit, reconciling conflicting benefits, and mapping existing real-time paradigms into benefit models. In sections 2 and 3, this paper considers the issue of defining benefit via motivating examples and by considering real-time computing efforts that have considered benefit. Section 4 provides a taxonomy that categorizes types of benefit.

2 The use of benefit in Real-Time applications

This section considers afresh what is meant by benefit, by examining the contexts of various real-time applications. Motivating examples are given from the domains of (1) multimedia, (2) air defense and (3) enterprise-level, asynchronous, cooperating real-time computer systems.

2.1 Multimedia

Multimedia applications can be designed according to the QoS Level model of soft real-time - each application is developed with a set of levels at which it can operate, each level having a particular set of resource requirements, a corresponding benefit, and an associated period (see [9] for details). Each application is accompanied by a specification, listing the levels and their resource needs, benefits, and periods. The QoS Levels chosen for a particular application are highly application-specific and reflect the particular soft real-time policy that the developer wishes to employ. A given application can soften one constraint multiple times to create a set of levels, or soften several simultaneously in different ways for the different levels. The only restriction is that the levels be ordered according to the resource used and the benefit provided. The levels can be implemented as parameters on the algorithms that implement the functionality of the application, or can consist of calls to completely different algorithms.

Desktop video playback is a non-critical real-time application. It has deadlines associated with the playback of individual frames of video, but failure to meet those deadlines results in lower user satisfaction rather than outright failure of the application or system (except in the most extreme cases). Furthermore, the amount by which user satisfaction is decreased as deadlines are missed is dependent upon the number of missed deadlines, the amount by which deadlines are missed, and the frequency with which they are missed. In other words, the benefit of video playback is a function of the degree to which its

deadlines are met. Other characteristics of video processing correlate directly with perceived user benefit including the size of the displayed image, the color depth, the resolution, the frame rate, compression losses, etc. Furthermore, there is a benefit associated with transitioning between different benefit levels. For example, while a video played at one display size may have one benefit, and the same video played at another display size may have a different benefit, the benefit achieved by alternating between those two sizes will not be the average of the two individual benefits; in fact, it will be lower then either of them. This dynamic benefit is independent of the static benefit achieved by the application and varies depending upon the characteristics of the particular application.

2.2 Air defense

A (partial) air defense subsystem can be modeled using three dynamic end-to-end execution paths: threat detection, engagement, and missile guidance. The threat detection path examines radar sensor data (radar tracks) and detects potential threats. The path consists of a radar sensor, a sensor data stream, a filtering program and an evaluation program. When a threat is detected and confirmed, the engagement path is activated, resulting in the firing of a missile to engage the threat. After a missile is in flight, the missile guidance path uses sensor data to track the threat, and issues guidance commands to the missile. The missile guidance path involves sensor hardware, software for filtering, software for evaluating and deciding, software for acting, and actuator hardware.

Threat detection is a sensor-data-stream-driven path, with desired end-to-end cycle latencies for evaluation of radar track data. Peak loads cannot always be known in advance for the threat detection path, since the maximum number of radar tracks can never be known. Furthermore, average loading of a path is often a meaningless notion, since the variability in the sensor data stream size is very large (it may consist of zero tracks, or it may consist of thousands of tracks). The path may fail to meet the desired timeliness Quality of Service in a particular cycle. However, the path must continue to process track data, even though desired end-to-end latencies may not be achieved in all cycles. Thus, there is benefit in performing threat detection even if one (or more) period deadline is missed. Detection of a threat track is very beneficial, and the earlier it is detected the higher the benefit. Detecting a threat when there is time to engage it has acceptable utility, detection at a time that is too late for engagement has unacceptable utility.

Engagement is driven by a stream of events sent by the situation assessment path. It uses inputs from sensors to determine which actions should be taken and how the actions should be performed, notifies actuators to start performing the actions, and informs the action guidance path that an action has been initiated. Typically, a timing objective is associated with the completion of the initiation sequence. The real-time QoS of this path has a higher priority than the real-time QoS of the continuous threat detection path. Thus, there is greater benefit in the aperiodic engagement of a threat in a timely manner than in meeting a deadline for the periodic threat detection path.

The missile guidance path of the air defense example is activated by a missile launch event. Once activated, the path periodically issues guidance commands to the missile until it detonates (the deactivation event). The required completion time for one iteration is dynamically determined, based on characteristics of the threat. If multiple threat engagements are active simultaneously, the threat engagement path is responsible for issuing guidance commands to all missiles that have been launched. The missile guidance path has two timeliness objectives: (1) cycle completion time: the duration of one iteration of the "monitor, plan, command" loop, and (2) action completion time (or deactivation time): the time by which the action must complete in order to succeed. It is more beneficial to perform the required processing before the action completion deadline than it is to meet the completion time requirement for each guidance cycle (although the two deadlines are certainly related). In other words, it is acceptable for the completion times of some cycles to violate the cycle deadline requirement, as long as the desired actions are successfully completed by the deactivation deadline.

In reality, it is very difficult, if not impossible to track an arbitrary number of tracks with any fixed system. Assuming that a non-trivial amount of computational resources are used to track a single target, as the number of targets is increased, the system will reach a point where it cannot track every target, even with adaptive resource management techniques. In such situations, benefit models are applicable. Identification of near targets yields higher benefit than identification of targets that are far away. Similarly, there is greater utility in classifying fast tracks than there is in the classification of slow tracks. Given flexibility in the fidelity and rate of tracking algorithms (which is often the case), one can use benefit functions to determine the frequency with which each target is tracked and to select the accuracy of tracking algorithms employed to process tracks. For example, one may need to track at a high frequency and with high accuracy in order to launch a missile, but low accuracy and low frequency may serves perfectly well otherwise.

2.3 Enterprise-level, asynchronous, cooperating Real-Time computer systems

Enterprises of the 21st century will increasingly consist of many autonomous systems that cooperate to perform a variety of missions (see Figure 1). The air defense example described in the preceding section is one small component found in modern military enterprises, such as the one depicted in Figure 2. Additionally, we have observed this structure in the domains of space systems and teams of autonomous mobile robots that cooperate to accomplish a task. In [3], the authors provide an autonomous vehicle example, which also illustrates this type of system. In such systems, mission priorities vary dynamically according to complex rules. Thus, the benefit of meeting a particular real-time requirement can vary significantly over the lifetime of the enterprise and is a function of many interrelated entities.

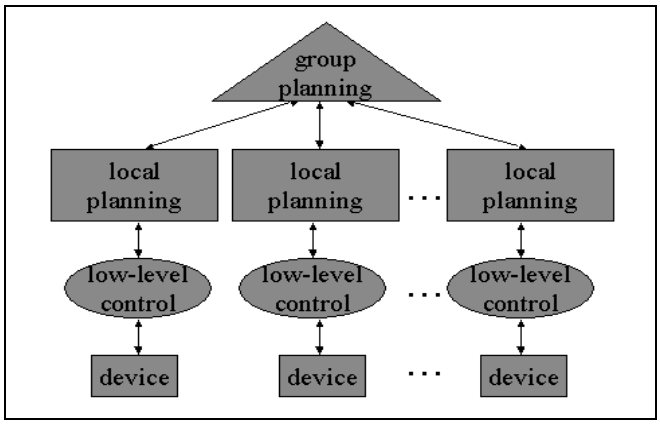

Figure 1: A hierarchical model of cooperating autonomous brokers

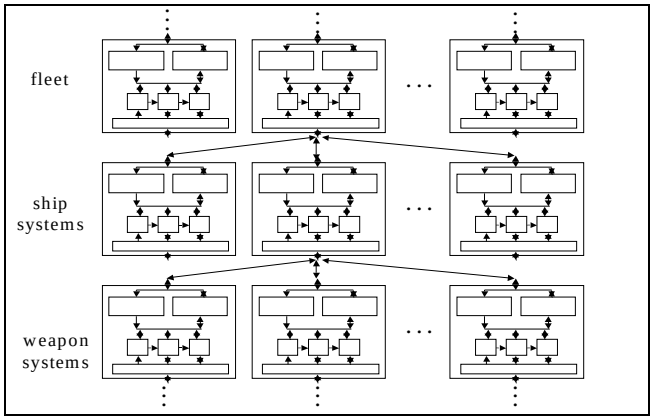

Figure 2: A hierarchical agent based navy system

3 Paradigms that incorporate benefit

The various ways in which the notions of benefit, utility and value have been employed in computing systems is surveyed in this section. The use of utility in the agent based computing paradigm is first presented. This is followed by discussions of three general classes of real-time computing paradigms that incorporate benefit.

3.1 The notion of utility in agent based computing

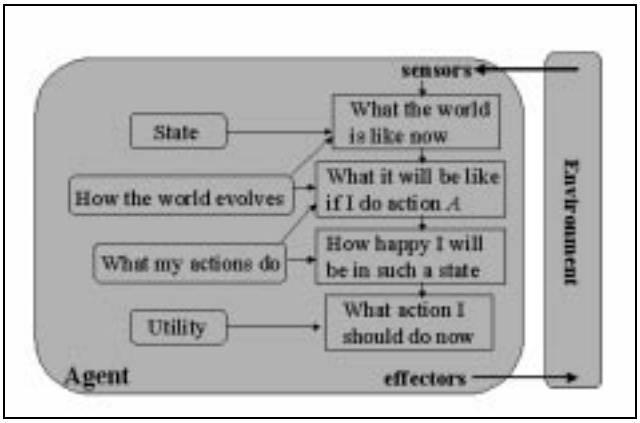

Figure 3: A complete utility-based agent (from [12])

Valuable insight into the concept of utility can be gained by looking outside of the real-time computing community to the agent based computing models. As described in [12], an agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through effectors; and a rational agent is one that "does the right thing." Figure 3 shows the overall structure of a rational agent that determines it course of action by applying utility theory [13, 14]. In [12] an *ideal rational agent* is defined as one that does whatever is expected to maximize it performance measure.

While such an agent is desirable to employ, it is, unfortunately, not always possible to employ. Practical considerations such as computational complexity and space complexity of "ideal" algorithms are often prohibitive. Rather, the benefits of observing real-time objectives of acting upon the environment require that absolute perfection be abandoned for reasonable, timely responses.

3.2 Priority, importance and criticality

The notion of priority (sometimes referred to as importance or criticality) is the simplest method for encoding fixed benefit relationships. With this scheme, a task is assigned a priority that is higher than the priority of

tasks whose real-time performance yields lower benefit. Such an approach can be taken by carefully assigning priorities to real-time tasks. A more sophisticated approach involves combining deadlines and criticalness as described in [6].

Also in the same class of approaches is the rate monotonic approach [2], wherein priority is determined by period or rate of execution. For aperiodics, timing requirements are assumed to be "soft." In other words, the paradigm assumes that there is greater benefit in meeting a timing constraint of a periodic than of an aperiodic, and that benefit is highly correlated with execution rates for periodics. This may not correspond to the utility perceived by the users (e.g., see the air defense example above). Furthermore, in RMA it is assumed that all on-time completion times for hard real-time tasks have equal utility, which may not always be the case (see [1,6]).

3.3 The use of benefit in scheduling Real-Time systems

It is important to distinguish between time invariant benefit and time varying benefit. In [1], Jensen points out the following: "In fixed priority scheduler systems, deadline management is attempted by assigning a high fixed priority to processes with 'important' deadlines, disregarding the resulting impact to less 'important' deadlines. During the testing period, these priorities are (usually manually) adjusted until the system implementer is convinced that the system 'works.' This approach can work only for relatively simple systems, since the fixed priorities do not reflect any time varying value of the computations with respect to the problem being solved, nor do they reflect the fact that there are many schedulable sets of process deadlines that cannot be scheduled with fixed priorities. In addition, …extremely difficult to determine reasonable priorities, since, typically, each individual subsystem implementer feels that his or her module is of high importance to the system…results in real-time systems with marginal performance…also in extremely fragile systems in the presence of changing requirements."

To address these issues, Jensen et al. [1] were the originators of the explicit use of benefit in real-time computing and developed a paradigm supporting time variant benefit. In [1], they describe their perspective as follows:

- "It is our thesis that the most important difference between the real-time operating system and other computer systems is that in a real-time system, the completion of a process has a value to the system which varies with time."
- "…the completion of a process or a set of processes has a value to the system which can be expressed as a function of time."
- "Applying this model…we measure a number of well-known scheduling algorithms with respect to the resulting total system values."
- "This approach to measuring the process scheduling effectiveness is a first step in a longer term effort to produce a scheduler which will explicitly schedule real-time processes in such a way that their execution times maximize their collective value to the system…"
- "The value of a real process completion is well modeled by a step function only in those few cases where there is no longer any value in completing the process after its deadline…In many actual systems, the value of completing a computation after its deadline may rapidly decay, but remain positive…or completing a computation before its deadline may be more or less desirable than completing it exactly at its deadline."
- "Determining an appropriate value for a given process consists not only of simply assigning a priority, but rather of defining the system value of completing it at any time."
- "The use of value functions allows us to describe both hard and soft real-time environments, and, in particular, allows us to evaluate systems which mix deadlines of both types in a single system."

Other researchers have built on this early work. In [4], the authors acknowledge that "A generic way to provide multivalent description of tasks is to characterize each task by a time value function. This approach can be viewed as a complementary paradigm to the bivalent deadline-driven paradigm, especially in the case of overload." Their contribution is an $O(n^3)$ scheduling heuristic for maximizing tasks' contributions. Other scheduling approaches can be found in the Ph.D. theses of Jensen's students [5, 7], in DEC's Alpha and Mach kernel's [5] and in the work of Buttazzo [15]. The meaning and role of value in scheduling flexible real-time systems is discussed by Burns et al. in [3] and by Jensen in [10].

3.4 Flexible soft Real-Time processing

In [9], Brandt presents an innovative approach that incorporates the use of benefit for flexible soft real-time systems. In traditional real-time system design, a single algorithmic solution is selected based on the features of the hardware platform and the operational requirements of the project. Worst- case estimates of resource requirements are used in order to guarantee that the system requirements will be met. However, most real-time problems admit of a variety of algorithmic alternatives differing in their processing requirements and in the quality of their results. By deferring the decision of which algorithm to execute until run-time and to dynamically select the algorithm online based on the available resources and the importance of each individual task, less

resource intensive algorithms can be executed when necessary and higher fidelity algorithms can be executed when possible. The result is that resources can be more effectively applied where needed, more data can be processed overall, and an overall higher benefit can be achieved from the system.

In support of this type of adaptive processing Brandt developed the Dynamic Quality of Service Resource Management (DQM) system, an adaptive soft real-time system that dynamically selects an appropriate algorithm for each running application based on current system state and application needs. Given a specification of the algorithmic alternatives available and the corresponding resource cost and output quality, the DQM chooses the current best algorithm for each application. The current implementation uses a centralized middleware DQM implemented on top of a general-purpose operating system (Figure 4).

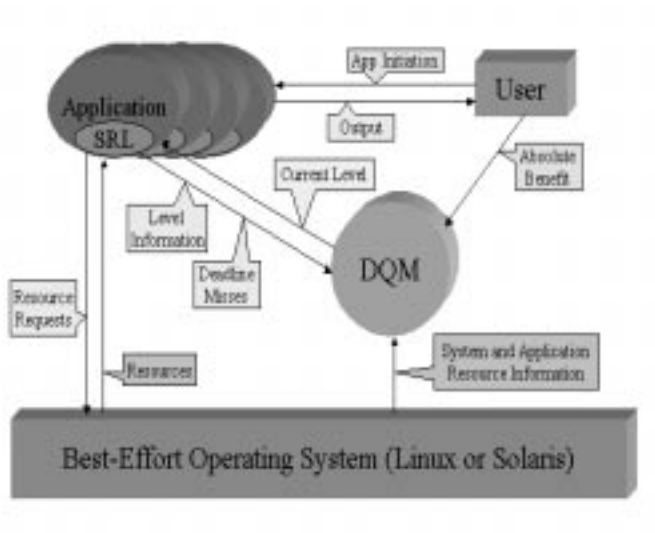

Figure 4: The DQM System

The DQM is based on a technique called QoS Levels, a method for providing soft real-time processing on a general-purpose operating system and as well as a model for adaptive applications to dynamically modify their resource usage. QoS Levels are a policy-independent soft real-time mechanism that allows each application to specify and implement its own dynamic soft real-time policy.

4 Categorizing types of benefit

The systems discussed above all use benefit to capture the importance of different aspects of the computations being executed. However, they vary in terms of how they define and use benefit. In particular, they vary in terms of the level of abstraction, the perspective from which benefit is determined, whether they consider benefit to be a static or dynamic quantity, and whether it is viewed simply as an input to the system, or as both an input and an output. These differences have a significant effect on the details of the system in question and thus merit further consideration. In particular, an understanding of these details is necessary in order to understand the use of benefit in each of these systems and in developing an overall theory of the use of benefit in real-time systems.

4.1 Different levels of abstraction (time/value, resource/value, etc.)

Benefit in real-time systems can refer to one of several different levels of abstraction, ranging from a very low-level view of the system in terms of individual periods of running processes and their deadlines to a very high-level view of the system in terms of applications, user goals and relative resource allocations. Importantly, the level of abstraction influences how benefit is defined, specified, calculated, employed in decision-making, and measured.

4.1.1 Benefit as a function of time

At the lowest level of abstraction, benefit is calculated in terms of individual periods of running processes and their deadlines. Specifically, benefit is defined as the value of a particular iteration of a periodic process as a function of its completion time as it relates to the deadline for that period, as in Jensen's system [1]. At this level of abstraction benefit is specified as a time-value function showing the value of a given iteration of a periodic process as a function of its completion time relative to its deadline. These time-value functions can be used to determine which process to execute at any given time to maximize the overall benefit of the system and to calculate the benefit resulting from any particular hypothetical or actual execution order.

4.1.2 Benefit as a function of time and resource allocation

At a slightly higher level of abstraction benefit is calculated as a function of the completion time as it relates to the release time or deadline for that period but controlled by a higher level resource allocation mechanism, as in DeSiDeRaTa [11], SMART [16], and Rialto [17]. In this case benefit is specified as a direct linear function of time and may or may not be used in the actual scheduling decisions as any scheduler that minimizes wait time will also tend to maximize benefit. Benefit is controlled by managing resource allocations so as to effect the behavior of the scheduler.

At this level of abstraction, benefit is calculated directly from the wait times or amount by which each deadline is missed and is used as an input to resource allocation decisions at a higher level. Changing resource allocations affect either the operation of the scheduler or the location of various processes (in a multiprocessor system), thereby changing the resources allocated to each process.

4.1.3 Benefit as a function of output quality and resource allocation

At an even higher level of abstraction benefit can refer to the overall quality of the output of a process given a specific level of resources allocated [9], without reference to any notion of time. In this case, benefit is a direct reflection of the output quality of the process and is not necessarily a function of the actual completion times. Generally, benefit is a specification of the output quality achieved by running the application with a given resource allocation. The differing quality of the output is a result of running different algorithm at each different resource allocation, and not as a result of missing deadlines by greater or lesser amounts. In particular, missed deadlines are an exceptional condition used to indicate that an application is not satisfactorily executing its algorithm within the allocated resources and triggers a change in algorithm or resource allocation to correct the problem. In this case, benefit is measured as a function of the level at which each application actually executes.

Interestingly, at the lower levels of abstraction the measured benefit of the system will be exactly as calculated, while at higher levels, the measured benefit may differ from the calculated benefit because the connection between the benefit and the actual low-level execution of the scheduler is less direct. It should be added that other levels of abstraction are not considered in this discussion they are by no means impossible.

4.2 Perspective

In considering benefit it is also important to consider the perspective from which the benefit is being calculated as this differs in different systems and affects the way that benefit is both defined and used. This is closely related to the level of abstraction but is more or less orthogonal. The four perspectives that typically occur in real-time systems are the thread perspective, application perspective, system perspective, and user perspective. These different perspectives may agree or compete, depending on the circumstances.

4.2.1 Thread perspective

The thread perspective reflects the needs of each individual unit of execution in the system. Systems dealing with a lower level of abstraction also tend to manage benefit at this level and calculate benefit from this perspective. From this perspective, each thread is greedy and benefit is accrued on a per-thread basis. Furthermore, from this perspective benefit is a zero-sum game inasmuch as raising the benefit of one thread (by giving it more resources, scheduling it earlier, etc.) necessarily means taking it from another thread.

From the thread perspective, benefit is generally a function of the number, frequency, or amount by which deadlines are missed. Maximizing thread benefit generally means minimizing the number of missed deadlines, the amount by which deadlines are missed, or the frequency of missed deadlines.

4.2.2 Application perspective

The application perspective reflects the needs of the applications in the system, where an application may be comprised of more than one process. Each application is fundamentally greedy, requiring the resources it needs and generally providing better (or at least no worse) service, and thus higher benefit, with increasing resources. In hard and firm real-time systems the benefit is relatively fixed as long as the application deadlines are being met. With soft real-time the benefit may continue to increase with increasing resources up to some maximum limit, even when all deadlines are already being met [9, 18]. Even with hard and firm real-time applications, more resources can always be considered better in the sense that with greater resources each application is in some sense less likely to miss a deadline through data dependent variations in processing time or other system fluctuations [11].

While individual threads may cooperate to achieve the goals of the applications, application benefit can remain constant despite changes in thread-level execution. In particular, application benefit is not merely the sum of a theoretical lower-level benefit of the threads implementing the application. In fact, in systems dealing with application benefit, benefit of individual threads is generally not considered or calculated.

From the application perspective, benefit is generally a function of the amount of resources allocated to each application. Applications receiving more resources produce higher benefit while applications receiving less produce lower benefit. With applications consisting of only a single thread, the thread and application perspective may be equivalent inasmuch a thread receiving more resources will make more of its deadlines or miss its deadlines by smaller amounts.

4.2.3 System perspective

The system perspective reflects the needs of the system. Maximizing system benefit generally means maximizing

the summed benefit of the currently running applications (normalized by the relative importance of those applications) and, to a lesser degree, to maintain high levels of system utilization, possibly by running background tasks when possible (which, while non-real-time, may have benefit of their own, particularly to the user).

System benefit, while relatively straightforward to calculate, involves several inherent conflicts. First, because of the relative differences in resource requirements of and benefit provided by the different applications in the system, maximizing system benefit often requires difficult tradeoffs among the different running applications, particularly when the combined resource requirements of all of the applications exceeds the available resources in the system. In this case it is necessarily the case that from the perspective of some applications, application benefit will be less than maximal. Furthermore, it can also be the situation that the resource allocation or scheduling decisions that maximize system benefit can be completely unsatisfactory to the user (discussed more in the next section).

The second conflict is inherent in the fact that higher utilization produces higher benefit, but also results in a situation where there are less free resources available. In many soft real-time systems the actual application resource usage varies and is often characterized in terms of average-case usage instead of worst-case usage. When the resource usage of such an application is greater than its allocation, it will meet its deadline if resources are available, either because of resources committed to another application but unused because the other application used less than its allocation, or because of available uncommitted resources in the system. However, if no unused or uncommitted resources are available, the application will miss its deadline. Because any application can use all of the resources that it has been allocated, the only way to guarantee that free resources will be available for an application that exceeds its average-case allocation is to keep some slack resources uncommitted to any application, and fewer free resources can cause more frequent deadline misses and thus lower application and system benefit. However, the cost of keeping some resources unallocated is that these resources could have brought an increase in benefit if they had been allocated to some application.

4.2.4 User perspective

The user perspective reflects the needs of the user or users of the system. This is generally the sum of the thread or application benefits for all of the applications that the user is running. While the system goal is usually to maximize system benefit, this may not always reflect the needs of the user, especially when there is more than one user. Furthermore, user goals and system goals can conflict in certain situations. These conflicts can be managed, but they must first be identified.

Application	Level	Resources	Benefit
A	1	25%	1
	2	50%	2
B	1	25%	1
	2	50%	2
C	1	50%	1

Table 1

For example, consider a situation with three applications, as depicted in Table 1. In this example there are three applications, two of which (A and B) can run at two different QoS levels, taking 25% and 50% of the available resources and providing normalized benefits of 1 and 2. The third application, C, can run at a single level requiring 50% of the resources and providing a benefit of 1. In this situation, the maximum system benefit attainable is 4, achieved by running applications A and B at level 2. However, given that the user ran application C, it is reasonable for the user to expect that application C will run, if only at it's lowest level (it's only level, in this case). So the user's preference might be for applications A and B to run at level 1. And application C to run at level 1. This results in a total benefit of 3, lower than the system could otherwise achieve, but maximizing user benefit. Other examples are possible.

A second example where user benefit and system benefit fail to agree is in situations of changing resources or resource requirements, either because applications are entering or leaving the system or because applications are varying their resource requirements and resource usage. In this case, available resources vary and in an attempt to maximize system benefit the system may want to change the levels of various applications. However, the change itself can result in lower user benefit. This is discussed further in the next section.

4.3 Static/stateless vs. dynamic/meta/stateful

Most systems treat benefit as a static concept; benefit is determined based on a calculation of the state of the system and is entirely stateless. In other words, each calculation is independent of any previous calculation and is a simply statement about the current state of the system. However, in some situations user benefit can have a dynamic component. More specifically, user benefit at a given time can be a function of the current state of the system and one or more previous states of the system.

This is best illustrated with an example. Consider a video display system that is running steadily at 15 frames per second. This is imperfect since the viewer will see some flicker, but it is good enough to watch a video on a computer. Now consider the situation where the same application frequently alternates between 15 and 30 frames per second (by showing more frames, thus technically improving the quality of the video stream). 30 frames per second will produce a better video stream, but alternating between 15 and 30 frames per second may itself very be distracting, with the end result that the overall quality of the video has decreased as perceived by the user. Thus the benefit of running at 30 frames per second is in part dependent on the state of the system just before we ran at 30 frames per second. If it was previously running at 30 frames per second, then the benefit is high, whereas if it was previously running at 15 frames per second, then, at least temporarily, the benefit is low. The same argument can be made for a variety of applications and quality metrics - consider latency in audio, for example, where different latencies produce equal benefit, but varying latency produces lower benefit.

We conclude that benefit cannot always be determined based on a static calculation at a given instant but must, at least in some instances, be a function of the current and previous states.

5 Conclusions and future work

While the future of real-time computing research and practice will likely exploit the utility of benefit-based models, there are many issues which must be resolved at present. This paper provides a valuable perspective of some of the issues, by providing motivating examples for which the traditional hard/firm/soft real-time paradigms are inadequate, and by summarizing the previous work on benefit-based computing models. It also presents issues that should be considered in order to increase the usefulness of benefit-based real-time computing models. The problem of unifying the various types of benefit, including those from the application perspective, remains open. Interesting sub-problems include (1) deriving benefit specifications from real-time constraints and (2) resource allocation decision procedures that consider all aspects of benefit.

6 References

[1] E. D. Jensen, C. D. Locke and H. Tokuda, "A Time-Driven Scheduling Model for Real-Time Operating Systems," *Proceedings of the Real-Time System Symposium*, 112-122, IEEE CS Press, 1985.

[2] C.L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, **20**, 1973, 46-61.

[3] A. Burns, D. Prasad, A. Bondavalli et al., "The Meaning and Role of Value in Scheduling Flexible Real-Time Systems," *Journal of Systems Architecture*, 46 (2000), 305-325, Elsevier, 2000.

[4] K. Chen and P. Muhlethaler, "A Scheduling Algorithm for Tasks Described by Time Value Function," *Journal of Real-Time Systems*, 10, 293-312, Kluwer, 1996.

[5] R. K. Clark, E. D. Jensen, F. D. Reynolds, "An Architectural Overview of the Alpha Real-Time Distributed Kernel," Proceedings of the USENIX Workshop on Microkernels and Other Kernel Architectures, April 1992.

[6] S. R. Biyabani, J. A. Stankovic and K. Ramamritham, "The Integration of Deadline and Criticalness in Hard Real-Time Scheduling," IEEE publication CH2618-7/88/0000/0152, 1998.

[7] C. D. Locke, "Best-Effort Decision Making for Real-Time Scheduling," Ph.D. Thesis, CMU-CS-86-134, Department of Computer Science, Carnegie Mellon University, 1986.

[8] R. K. Clark, "Scheduling Dependent Real-Time Activities," Ph.D. Thesis, CMU-CS-90-155, Department of Computer Science, Carnegie Mellon University, 1990.

[9] Scott A. Brandt and Gary J. Nutt, "Flexible Soft Real-Time Processing in Middleware," *Journal of Real-Time Systems*, Kluwer, 2001.

[10] E. D. Jensen, "Asynchronous Decentralized Realtime Computer Systems," *in Real-Time Computing – NATO Advanced Study Institute on Real-Time Computing*, NATO ASI Series, Series F, Computer and Systems Sciences, v.127, pages 347-371, Springer-Verlag, 1994.

[11] L. R. Welch, B. Ravindran, B. Shirazi and C. Bruggeman, "Specification and analysis of dynamic, distributed real-time systems," in *Proceedings of the 19th IEEE Real-Time Systems Symposium*, 72-81, IEEE Computer Society Press, 1998.

[12] S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach," Prentice Hall, 1995.

[13] E.J. Horvitz, J.S. Breese and M. Henrion, "Decision Theory in Expert Systems and Artificial Intelligence," *International Journal of Approximate Reasoning*, 2:247-302.

[14] M.P. Wellman, "Reasoning About Preference Models," Technical Report MIT/LCS/TR-340, Laboratory for Computer Science, MIT, Cambridge, MA.

[15] Giorgio Buttazzo, Marco Spuri and Fabrizio Sensini, "Value vs. Deadline Scheduling in Overload Conditions," in *Proceedings of the 19th IEEE Real-Time Systems Symposium*, IEEE Computer Society Press, 1998.

[16] J. Nieh and M. Lam, "The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications", *Proceedings of the 16 th ACM Symposium on Operating Systems Principle*s, 1997.

[17] M. Jones, D. Rosu, M. Rosu, "CPU Reservations and Time Constraints: Efficient Predictable Scheduling of Independent Activities", *Proceedings of the 16 th ACM Symposium on Operating Systems Principle*s, 1997.

[18] T. Abdelzaher, E. Atkins, and K. Shin, "QoS Negotiation in Real-Time Systems and its Application to Automated Flight Control," *Proceedings of the 3 rd IEEE Real-Time Technology and Applications Symposiu*m, June 1997.