A Framework for Using Benefit Functions In Complex Real Time Systems

David Andrews
Electrical Eng. & Computer Science
ITTC
University of Kansas
Lawrence, KS 66045
dandrews@ittc.ukans.edu

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

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

Abstract

Researchers are currently investigating applying benefit, or utility functions for allocating resources in limited, soft real time systems [1,2,3]. While the future of real-time computing research and practice will likely exploit the utility of benefit-based models, this will not occur until a suitable system level benefit framework has been defined. This paper provides an overview of ongoing work to develop such an initial framework that can formalize the use of benefit functions in complex real time systems. It is easily shown that this framework can support traditional hard/firm/soft real-time paradigms as well as support newer proposed models for future real time operation [5,6]. We also show that our proposed framework unifies composition of benefit values derived from heterogeneous perspectives, including those derived from the application perspective.

1: Introduction

Benefit models have been shown to be effective [4][7][9] for the integration of small numbers of processes but have not been applied to the integration of large numbers of processes in complex relationships. However, the abstract notion of benefit extends to complex systems with many layers and many different types of benefit [10][8]. These include application notions of benefit, single system notions of benefit involving resource tradeoffs within that system, and aggregate system notions of benefit involving tradeoffs across systems and based on complex relationships between the systems and between the applications running on those systems. In such complex systems, one cannot simply maximize the benefit of each individual application and system and hope to achieve optimal system performance.

Without loss of generality our approach interprets the traditional concepts of "hard", "soft", and "firm" within an algebraic framework that supports composition of different levels of benefit abstraction. Our framework encourages the premise that at any instant of time, benefit within a complex enterprise system is a function of many interrelated entities that vary according to complex rules and interactions among different subsystems.

The different levels of abstraction at which a calculated benefit can affect aggregate system behavior is easily understood. At the lowest level of abstraction benefit can be viewed as a function of time, where applications have varying benefit based on the time instant at which they complete their processing relative to their deadlines. However, benefit may also be viewed as a function of the completion times and the resources allocated to the processes, as a function of resources allocated and of the quality of the output produced at that resource level, or solely based on the quality, frequency, or other properties of the output.

The perspective from which benefit is calculated is also important. Benefit may be calculated from the application perspective. It may also be calculated from the single system perspective, from which summed benefit of the set of applications is the important measure, but spare

resources may also be important. In this case, the spare resources themselves can be characterized with a benefit function and included in the calculations [8]. From a complex system perspective, however, there are many tradeoffs to be made in terms of CPU usage, network bandwidth, individual system capabilities, etc., and simply summing the benefit of the individual systems will not capture all that matters in such a system. Finally, there is also a notion of end-user benefit in many systems, and what the user actually cares about may differ markedly from what has been specified by the developer.

Because they differ in important respects and because many of these can be expected in complex real-time systems of the future, we believe that it will not be possible to maximize benefit through a simple optimization as is done in current soft real-time systems. Accordingly, we are developing a framework for describing and composing benefit with the goal of being able to use benefit in such highly complex real-time systems.

2: Benefit Function Definition

We define a value of benefit associated with attempting to meet some specified objective as a point in the interval:

$b_i = [l_i, 0, u_i]$ such that $l_i \leq 0$ and $0 \leq u_i$

For observed objective i, $b_i = 0$ denotes a quiescent state associated with the objective exactly meeting a specified requirement. If the benefit value $b_i > 0$, this denotes that the objective is not only meeting it's requirement, but is also providing additional capability beyond it's minimum. If the value $b_i < 0$, this denotes the objective has dropped below it's minimum requirement, but is still returning a diminished benefit value. For a traditional "hard" real time objective, a limitation on the acceptable value of benefit can be interpreted as interval [0, . . +∞]. For "soft" real time objectives, the benefit value b_i can assume any value within the interval [-∞, . . 0 . .+ ∞]. The lower and upper limits, l_i, u_i respectively, quantify limits for benefit optimization calculations. L_i represents the lowest allowable value of benefit that the objective is allowed to return. U_i represents a point at which the value of benefit returned is at the upper limit and any additional benefit is superfluous. Optimizing benefit is viewed as conditionally maximizing the value of the benefit vector subject to each benefit value being in $l_i \leq b_i \leq u_i$.

As a simple example, we can optimize benefit by summing the individual values as:

$\text{Max } B = \text{Max } \{ \Sigma b_I \mid l_i \leq b_i \leq u_i \}$

Where our search space is bounded by

$\Sigma l_i \leq \Sigma b_I \leq \Sigma u_i$.

More complex benefit calculation functions based on heuristics appropriate for a specific system can also be used. Using the definitions above, we can compose a benefit vector $B = <b_0, b_1, b_2, \ldots b_n>$

with defined associated limit vectors

$L = <l_0, l_1, \ldots l_n>$ and $U = <u_0, u_1, \ldots u_n>$.

Our definitions support hierarchical composition of benefit vectors, and allow abstract composition and optimizations to be performed.

It is also worth noting that benefit functions can be used for analysis during the initial system design by monitoring the systems ability to meet each specified lower level of benefit.

2.1: System Definitions Using Benefit

We provide the following definitions for our benefit framework.

Defintion 1. A "Hard" real time system is defined as:

$H_{sys} = \{ \forall i \mid l_i = 0 \}$

Simply stated, a hard real time system is a system such that the minimum acceptable levels of benefit are not allowed to be negative.

Defintion 2. A system is quiescent if:

$\text{Quiescent}_{sys} = \{ \forall i \mid b_i = 0 \}$

A system is in a quiescent, or steady state, when all measurable benefit is exactly returning the expected benefit value.

Definition 3. A system's performance is formally correct if:

$\forall i \mid b_i \geq l_i$ for all time t.

This definition relates the system's ability to always meet minimal acceptable levels of performance.

Definition 4. A system is under-constrained if:

$\Sigma b_i \geq \Sigma l_i$ and $\forall i \ b_i \geq l_i$.

This denotes that the system is meeting each individual benefit minimum, and the aggregate is providing additional benefit beyond the quiescent capabilities for which it was designed. This is characteristic of a system with excess capacity.

Defintion 5. A system is over-constrained if:

$\Sigma b_i \leq 0$ and $\forall i \ b_i \geq l_i$

Note that a system may be over-constrained, but still adhere to a formally correct performance behavior.

Lemma 1. A system with all benefit values classified as "hard" cannot be over-constrained and still formally meet it's performance requirements.

3: Application Example

The utility of benefit functions allows their use within a system to reflect the specific operational paradigm of the system without loss of generality. Consider the following example of a function that acts as a "bridge" node within a distributed system. This function is a periodic task that attempts to bridge as many messages as possible between two messaging channels within a given time limit. Once the task has reached the time limit, a benefit value is calculated based on the number of messages it successfully bridged. This capability is obviously dependent on the availability and performance of the interconnection networks, and may vary with time.

```
Function Bridge
Benefit = 0;
while(true){
        cnt = 0;
        while(current_time < limit){
          buffer = receive(ch1)
          send(ch2,buffer)
          cnt += 1;
        }
        if (cnt ==max) IPC.benefit = 0;
        elseif(cnt < max) IPC.benefit = -Δ;
        else IPC.benefit = +Δ;
        next_time = limit + interval;
        suspend(next_time);
}
```

In this example, each process defines a benefit Δ, representing the relative benefit to the system for over or under performing. Note that in this example, the function is not allowed to change the lower limit of acceptable performance. Instead, the lower limit is set by the system at initialization, and is maintained by the separate IPC benefit monitor as shown below. In this mode, the tasks are passive, and only report achieved benefit. Resources are allocated by the IPC monitor based on measured benefit from all functions performing messaging operations. The specific allocation scheme is dependent on the system.

```
_IPC_Performance_Monitor
If(benefit[task] < l[task]){
        if(excess > Δ){
                resource[task] += Δ;
                excess -= Δ;
        }
        else
                task_lowest = f(greatest_excess);
                resource[task_lowest] -= Δ;
                resource[task] += Δ;
        }
elseif(benefit[task] > l[task])
        resource[task] -=Δ;
        excess += Δ;
}
else(benefit[task] == l[task]);
```

In this simple example, the IPC Performance Monitor maintains a variable that represents the current excess capacity of the resources, and optimizes the excess capacity. If a task reports additional benefit, the monitor will decrease the resource and increase capacity. This simple algorithm optimizes the systems ability to meet minimal requirements and respond to dynamic fluctuations in requests.

This example shows a simple linear combination of benefits. The advantage of using this approach allows the application of more complex and system specific methodologies for combining and calculating individual and system benefits.

4: Conclusion

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 an initial framework to formalize the use of benefit functions in complex real time systems. It is easily shown that this framework can support the traditional hard/firm/soft real-time paradigms as well as supporting new models for real time operation. We provided a framework that unifies the various types of benefit, including those from the application perspective.

1 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 Systems Symposium*, 112-122, IEEE CS Press, 1985.

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

[3] 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.

[4] 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.

[5] Edward Lee, "What's Ahead for Embedded Software?", IEEE Computer, Sept. 2000, pp. 18-26

[6] Jason Hill, Robert Szewczyk, Alex Woo, Seth Hollar, David Culler, Kristofer Pister, "System Architecture Directions for Networked Sensors", Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX) Nov. 2000, pp. 93-104

[7] J. Hansen, J. Lehoczky and R. Rajkumar, "Optimization of Quality of Service in Dynamic Systems", Proceedings of the 9th International Workshop on Parallel and Distributed Real Time Systems, April 2001

[8] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, J. Hansen, "A Scalable Solution of the Multi-Resource QoS Problem", Proceedings of the IEEE Real-Time Systems Symposium, Dec. 1999

[9] 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.

[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.