The Meaning and Role of Value in Scheduling Flexible Real-Time Systems

A. Burns, D. Prasad
A. Bondavalli, F. Di Giandomenico
K. Ramamritham, J. Stankovic
L. Strigini

Computer Science Report No. CS-98-29
August 1998

The Meaning and Role of Value in Scheduling Flexible Real-Time Systems[*]

A. Burns and D. Prasad
Real-Time Systems Research Group
Department of Computer Science
University of York, UK

A. Bondavalli and F. Di Giandomenico
CNR, Pisa, Italy

K. Ramamritham
Department of Computer Science
University of Massachusetts, USA

J. Stankovic
Department of Computer Science
University of Virginia
Charlottesville, VA USA

L. Strigini
Centre for Software Reliability
City University, UK

Abstract

The real-time community is devoting considerable attention to flexible scheduling and adaptive systems. One popular means of increasing the flexibility, and hence effectiveness, of real-time systems is to use value-based scheduling. It is surprising however, how little attention has been devoted, in the scheduling field, to the actual assignment of value. This paper deals with value assignment and presents a framework for undertaking value-based scheduling and advises on the different methods that are available. A distinction is made between ordinal and cardinal value functions. Appropriate techniques from utility theory are reviewed. An approach based on constant value modes is introduced and evaluated via a case example.

[*]The work reported in this paper was undertaken as part of the AFERT Project (041) supported by a joint EU US funding arrangement

1 Introduction

The specification and design of any computer-based system (real-time or not) inevitably involves trade-offs and compromises. Not all possible services (to use a general term) can be supported. And those services that are implemented will not all have maximum functionality.

With most current real-time systems, the implementation phase has little flexibility. A fixed set of 'hard' services must be mapped on to the available resources, and pre run-time checks must be made to ensure that all timing constraints (typically, deadlines) are satisfied. This, of course, requires a predictable (bounded) model of the environment's impact on the computer system.

1.1 Need for Dynamic Scheduling

In this paper we attempt to increase the flexibility of real-time systems by allowing certain decisions about the system's behaviour to be made at run-time. This will require some form of dynamic scheduling. The notion of 'value' (of a service or sub-service) will be used to control the run-time decision process. Although certain aspects of value-based scheduling have been addressed before [21, 4, 1, 38, 7, 15], no systematic study has yet been published. In this paper we attempt to lay the foundation for the systematic approach that is needed.

The motivation for incorporating run-time decisions is largely based on two observations about the behaviour of typical static schedules.

- They use resources inefficiently. As there are sufficient resources to cope with the maximum possible load on the system (i.e., worst-case execution times, worst possible phasings, and worst-case arrival of sporadic work) the average resource utilization is low. Hence, there is considerable scope for value-added computation.

- They react inflexibly to failures and overloads. Although the static scheme may be able to cope with certain failures (as defined in its failure model), once the system moves outside its failure model or load hypothesis then no level of service can be depended upon. Hence, it is desirable to allow graceful degradation (of service) when resources are scarce.

The need to support value-added computation and graceful degradation is necessarily complicated by the reality that not all services have equal utility. When an implementation is constrained to support all services (all of the time), this is not an issue. For run-time decision making, it becomes an almost insurmountable one (because of the excessive overheads it inevitably introduces, at least in the general case).

1.2 Need for Value-based Scheduling

To keep the terminology clear we shall use the term 'utility' to mean the actual benefit that accrues from the delivery of services. In general this measure will not be known with absolute certainty. The 'value' of a service will be some approximation for utility, used to influence the real-time scheduling behaviour of the computer system. During the development of any system many design trade-offs are made. The system designer(s) must articulate preferences between services (or groups of services). Such preferences may be static or vary depending on operating conditions. They are based upon an informed belief in the utility of the system as represented by its services.

Due to the complexity of run-time decision making most trade-offs are evaluated at design time. To postpone such decisions until run-time can only be worthwhile if:

- The decision becomes much more effective if it takes into account run-time data/conditions.

- The decision would lead to pessimistic resource usage if taken pre run-time.

To facilitate run-time decision making requires the preference relationships to be stored and searched. However, this is only feasible if the overheads involved in making such decisions do not overshadow the potential benefits. This issue is a critical one, many published results in dynamic scheduling have ignored run-time overheads and are therefore misleading. As Chen and Muhlethaler [16] discuss, such scheduling is invariably NP-hard.

There are clearly a number of ways of holding preference relationships. In this paper we consider the use of a value attribute assigned to services and subservices. We present a framework (in Section 3) for value-based scheduling, and use this, in Section 4, to evaluate a number of approaches that have been reported in the literature. Section 5 of the paper then returns to the key issue of assigning values so that they represent an adequate approximation of the designers' preferences. An extensive example is introduced in Section 2 and considered in detail in Section 6. Conclusions are presented in Section 7.

2 Value-Based Scheduling

In simple terms, value-based scheduling is a decision problem involving the choice of a collection of services to execute so the 'best possible' outcome ensues. At various decision points (at run-time) there are a set of services that are available for execution. Unfortunately there may not be enough resources to satisfy all services. And hence, a decision must be made. This decision may involve picking out the 'extra' services to support when resources are spare, or which services to sacrifice when resources are scarce. Note that the particular scheduling regime is not significant; it could involve recomputing a schedule (as in Spring [30]) or managing slack time (as in a priority base schedule [20, 9]).

2.1 Problem Statement

We assume a computational model in which a system is deemed to be composed of a set of *services* (possibly inter-dependent). Each service may be realised by one or more *alternatives* – typically only one of which is needed for any particular invocation of the service. The entire system is considered to run indefinitely and so each service has an unbounded number of invocations. Services are either periodic or sporadic and hence there is a bound on the number of possible invocations within any time interval. Let S be the set of services:

$$S = \{S_1, S_2, ..., S_n\}$$

Each service is composed of alternatives:

$$S_i = \left\{A_i^1, A_i^2, ..., A_i^{m_i}\right\}$$

Each alternative has the following attributes defined (there may be others):

T – Minimum inter-arrival time
D – Deadline
C – Worst case execution time
V – Some value attribute or function (which may be time-dependent or state-dependent)

It will usually be the case that T is defined at the service level, but schemes in which each periodic alternative has a distinct period are possible. The first three terms are assumed to be known pre run-time. By comparison, V may be dependent upon a number of run-time factors.

If a service (or an alternative of a service) completes it will have some intrinsic benefit to the environment of the computer system (its utility). Hence, utility must be interpreted as the property of the operating environment not of the computer system itself. Benefit may be direct (by producing an output into the environment) or indirect (for example health monitoring of the processing units). In general, the value approximation of a service's utility may depend upon:

1. the quality of the output produced (e.g., accuracy, precision, statistical significance)
2. the time at which the service completes (i.e., too early, acceptably early, optimally delivered, acceptably late, too late)
3. the history of previous invocations of this service
4. the condition of the environment
5. the state of the computer system (i.e., what other services are being provided)
6. the importance of the service (a rough classification would be fundamental and non-fundamental services, where fundamental services have importance not comparable with that of non-fundamental ones)
7. the completion probability of the service.

2.2 Considerations

Before considering the notion of value further, we consider the nature of the decision problem.

> **Assumption 1:** Although during specification and design, trade-offs have to be made between conflicting (or at least not compatible) requirements, we assume that value-based scheduling is concerned only with the allocation of limited resources.

Hence, in an ideal implementation all services would complete with the best alternative used in each case; there would be no need for run-time scheduling decision to be made, and value-based scheduling would not be needed. The reality that resources are limited (and may fail) means that either the number and quality of services must be curtailed or that dynamic scheduling must be employed.

In a static approach the software for a service is designed to have adequate quality, its temporal requirements are represented by a hard deadline on completion, the environment is considered to be unchanging (or perhaps changing between a small number of modes), and the state of the rest of the system is usually ignored. Clearly there are improvements that can be made on this static approach. Trade-offs between quality of service, resources needed to furnish this quality, and the optimality of completion time are possible in many (application specific) ways. The relationship between services may also be very subtle. In an aircraft, if one set of control surfaces is being managed optimally then another set of services may be less important (for now). However, following a failure in some service the 'value' on another service may have to be increased radically.

Computing an accurate 'value' of a service is, of course, only useful if some advantage is gained. In job-shop scheduling where the planning period is relatively long and deadlines are specified in terms of minutes, if not hours, then comprehensive value functions are probably feasible. However, for systems with millisecond deadlines we have to make the following assumption:

Assumption 2: The run-time evaluation of an accurate and comprehensive value parameter for all services (and alternatives within a service) is prohibitively time consuming.

This assumption is supported by the work of Tokuda *et.al.* [32] and Wendorf [36]. They highlighted the difficulty in combining single values, derived for each of the relevant measurements given earlier, to obtain an "overall value". A further dimension of complexity comes from noting that the measurements may not be known with full certainty. Hence, there would be some attribute of belief (or confidence) associated with them which could be formalised via a probability distribution but would again be time consuming to evaluate at run-time.

In addition to this observation it is necessary to make two further assumptions about the general value-based decision problem (again these assumptions are well supported by both experimental and theoretical work).

Assumption 3: The computing of an accurate assessment of the total available resources (before some deadline) is prohibitively time consuming.

Assumption 4: Choosing the optimal subset of available services so that value is maximised and resource usage is bounded to the known available level is prohibitively time consuming (in general it is NP-hard).

This difficulty is compounded by the fact that the services will have different deadlines (hence, the available resources are not constant). The services may even be attempting a value/time trade-off by having alternatives with different deadlines or time-dependent value functions.

Assumptions 2-4, taken together, clearly indicate that the overheads involved in undertaking value-based scheduling are of major importance. Optimal, or even near optimal, performance cannot be expected. Rather it must be emphasised that the objective in using value-based scheduling is to facilitate flexible application behaviour, and to provide significantly greater benefit than would be obtained from using a simple scheduling approach such as FIFO or EDF. In Section 3 we present a framework in which the overheads implicit in assumptions 2-4 can be kept to a manageable level. Before that, however, we present an example that points to the advantages in exploiting dynamic scheduling.

2.3 An Example Problem: An Autonomous Vehicle Controller

Future real-time systems for autonomous vehicle control will need to exhibit intelligent and adaptive behaviour in order to function in a highly dynamic and non-deterministic environment characterised by the unpredictable nature of other vehicles, obstructions, route information, weather and road conditions.

Further, the consequences of system failure due to faulty software, hardware or sensors may be catastrophic. Thus autonomous vehicle control systems must be resilient to such failures and be imbibed with the property of graceful degradation under conditions of overload. Finally, cost and space constraints dictate that the most effective use must be made of the limited processing and communications resources available.

To realise such complex systems, two potentially conflicting objectives must be met: First, safety critical and mission critical services must be guaranteed to provide results of a minimum acceptable quality and reliability by their deadlines. Second, the utility of the system, as determined by the frequency, timeliness, precision and confidence level of the results produced, must be maximised.

Autonomous vehicle control systems offer improved performance by virtue of increased fuel efficiency, reduced journey times and accident rates whilst freeing the driver to carry out other activities during the journey. Indeed, autonomous vehicles may provide unattended pickup and delivery services, without the need for human intervention.

The tasks a human driver performs may appear to be rather straight-forward. However, building a fully autonomous vehicle calls for substantial advances in both technology and infra-structure. The major difficulties in automating the driving task arise from the unpredictable and dynamic nature of the environment as well as the need for accurate and timely sensing of other vehicles and obstacles, vision and scene interpretation, real-time decision making, route and path planing, and finally, communication and co-operation with other vehicles.

The major projects in this area [14] include PROMETHEUS (PROgraM for European Traffic with Highest Efficiency and Unprecedented Safety), IVHS (Intelligent Vehicle Highway Systems), PATH (Partners for Advanced Transit and Highways), DRIVE (Dedicated Road Infrastructure for Vehicle safety in Europe) and SSVS (Super Smart Vehicle Systems).

System Description

The functionality of an autonomous vehicle control system is highly complex, with a wide range of timing requirements. Within a time frame of 10-100^{+} seconds, it must plan and re-plan routes to reach the chosen destination, optimising fuel and time usage and take account of information about traffic conditions. On a shorter time scale, (approximately 1 second), scene recognition, assessment of other vehicle movements and path planning are required to ensure that the vehicle can steer a safe course, within comfortable ride limits and carry out maneuvers such as overtaking or merging with other traffic. Within a still shorter time frame of perhaps 100 ms, collision avoidance algorithms need to sample sensor data, detect possible collisions with obstructions or other vehicles and initiate avoidance and/or braking.

Due to the safety critical nature of autonomous vehicle control systems, they need to be resilient to individual sensor and processor failures and continue to operate effectively, albeit at a degraded level, under overload conditions. The amount of fault-tolerance that can be incorporated is limited by cost considerations. However, the on-board system still needs to have active redundancy, so that in the event of a unit failure, the vehicle can at least operate in a degraded mode until it achieves a safe state. In the case of software fault-tolerance, there is greater scope for analytical redundancy, exception handling, reconfiguration and recovery procedures rather than N-Version programming.

An assessment of an autonomous vehicle control systems [8] identified eight subsystems (i.e., services) and with each a number of alternatives – see Table 1. This example will be returned to later in the paper.

3 A Framework for Value-Based Scheduling

We start with some definitions that are typical of those used in value-based scheduling.

The scheduling process is assumed to have two distinct policies: an *admissions* policy and a *dispatching* policy. A request for scheduling is made when an optional alternative 'arrives' in the system. It may be *rejected* or *guaranteed* by the admissions policy (if one exists). The dispatching policy will use some scheme which will result in the alternative either *completing* successfully or *failing* to meet its deadline.

An alternative is defined to be *rescindable* if it can be revoked at any time (even if guaranteed). An alternative is *weakly rescindable* if it can be revoked at any time prior to actually starting its

SERVICE GROUP	ALTERNATIVE
Collision Avoidance	Infra-red beam deflection (IRBD) RADAR Short Range Communication (SRC) Stereo Vision based vehicle Tracking (SVT)
Braking Control	Basic braking Anti lock braking (ABS) Load sensitive ABS
Engine Control	Basic control Increased precise computation(1) Increased precise computation(2)
Lateral Control (input processing)	Magnetic markers Line Following
Lateral Control	Proportional Integral Derivative (PID) Frequency Shape Linear Quadratic (FSLQ) Fuzzy Rule-Based (FRB) Neuro Contol (NC)
Path Finding	Basic data fusion Fuel optimisation Ride comfort Time optimisation
Route Planning	Middle level planning GPS global planning
Displays Control	Basic update rates Speed warnings Fuel consumption analysis

Table 1: Autonomous Vehicle Example: Services and Alternatives

execution. A service that is optional but either non-rescindable or weakly rescindable is often called a 0/1 service (it must never start unless it can be completed).

It is not the objective of this framework to define actual policies. Section 4 will, however, review a number of methods based upon the definitions given here. It will, nevertheless, be necessary to use the following definition within the framework:

> **Definition 1:** A request to run a service alternative will be in one of the following states: pending, rejected, guaranteed, guaranteed but rescindable, admitted, completed or failed.

An admissions policy may not perform a guarantee, hence the weaker state of 'admitted' (but not guaranteed). It should also be noted that, in general, a guaranteed request may still fail (it depends upon the actual policies in use and the stochastic nature of the scheduling parameters and possible system faults).

In the following discussion we first consider, in more detail, the mathematical underpinning to the use of value. We then introduce the notion of mode in order to structure value based scheduling and to limit the complexity identified in the previous section.

3.1 Utility and Value

The notion of utility goes back to the eighteenth century. Roberts [26] notes that it was first used by Bentham in 1789.

> By utility is meant that property in any object, whereby it tends to produce benefit, advantage, pleasure, good, or happiness (all this in the present case comes to the same thing), or (what comes again to the same thing) to prevent the happening of mischief, pain, evil, or unhappiness to the party whose interest is considered: if that party be the community in general, then the happiness of the community; if a particular individual, then the happiness of that individual.

A value function is able to compare two objects using a binary relation. Let 'More_Useful' be such a relation. Then:

$$S_1 \ More\_Useful \ S_2 \Rightarrow V(S_1) > V(S_2)$$

where V is the value function that assigns a real number to the service (it is an approximation of the service utility).

Definition 2: If V is defined for only a subset of S_i then it is a partially ordered value function.

Definition 3: If V is defined for all S_i then it is an ordinal value function.

An ordinal value function has ranked all services, and hence,

$$S_1 \ More\_Useful \ S_2 \Leftrightarrow V(S_1) > V(S_2)$$

An ordinal value function is sufficient for deciding which single alternative of a service is to be preferred. The decision process is straightforward:

- Remove from consideration the alternatives that are clearly not feasible (i.e., there are insufficient resources to complete the alternative's execution by its latest deadline).
- Choose the remaining alternative with the highest value (or highest value density – see Section 4.2).

Choosing between services adds additional complexity. Although a partial ordering may be extended to enable a preferred service from a choice of two to be made, in general we are attempting to choose the preferred subset of a number of services. To consider a simple example, is it better to support service S_x or services S_y *and* S_z? One could deal with this by defining a [value] function on the universe of possible combinations of services, so as to represent the preferences. This could then be mapped, in a consistent way, to a cardinal value function. Alternatively, values could be assigned to services and then checked to see if the assignments are in accordance with the preferences (or more realistically, partial preferences) of the designers (this issue is returned to in Section 5.1).

Definition 4: A value function that is ordinal and additive (over the set of services/alternatives) is called a cardinal value function.

So

$$V(S_a \text{ and } S_b) = V(S_a) + V(S_b)$$

If the value function is cardinal then, in theory, the maximum value for the available resources can be obtained.

Although a number of scheduling schemes have used a pure cardinal value function they do not adequately deal with related services or alternatives. Two different situations may arise. First a group of services, if they are *all* executed, may have a value greater than the sum of the individual services; e.g.,

$$V(S_a \text{ and } S_b) > V(S_a) + V(S_b)$$

Alternatively, similar services may yield less value; e.g.,

$$V(S_a \text{ and } S_b) < V(S_a) + V(S_b)$$

For example, S_a and S_b may incorporate diverse software (which increases reliability) but the added benefit may not be as much as, in effect, adding their values.

3.2 Moded Behaviour

To control the potential complexity arising from a dynamic value function we restrict the variability, in a service's, or alternative's, value, to just three causes:

- The state of the environment
- The state of the underlying computing resource
- The state of the application software

The latter cause embraces the history of the alternative's execution and the current state (accepted, rejected, guaranteed or completed) of other related services/alternatives.

The three components are modelled as *modes*:

$$MODE = [ENV\_MODE, COMP\_MODE, SYS\_MODE]$$

Definition 5: Within a mode a fixed set of services and alternatives are defined.

Definition 6: Within a mode the value of each alternative is constant.

As values are fixed they can be determined pre run-time (see Section 5). Increasing the number of modes allows more specific dynamic behaviours to be accommodated.

In most situations it is useful to distinguish between *major modes* and *minor modes.*

$$MAJOR\_MODE = [ENV\_MODE, COMP\_MODE]$$

$$MINOR\_MODE = [MAJOR\_MODE, SYS\_MODE]$$

So, for example, relation $(V(S_a \text{ and } S_b) > V(S_a) + V(S_b))$ would be modelled by a minor mode (representing, for instance, S_a having been guaranteed) in which the value of S_b is constant but less than in the minor mode in which S_a has either been rejected or has failed.

To cater for the requirement that some services have value incomparable with other services, the alternatives available in a particular mode are split into *groups* (i.e., group A, group B ...). An ordinal relationship exists between the groups (e.g any alternative in group B has greater value than any in group C – moreover it has greater value than *all* alternatives in group C).

Definition 7: Within a mode, the group A set of alternatives may be defined to be *mandatory* if all these alternatives must be guaranteed to meet their timing requirements.

The use of a pre run-time guarantee means that a mandatory set of alternatives is not subject to value-based scheduling. However a service may be mandatory in one mode but subject to value-based scheduling in another (this is illustrated in the case example).

Definition 8: Within a group a cardinal value function exists.

3.3 Summary

The proposed framework has a number of key notions. First the use of 'mode' to define areas of operation in which 'value' can be assumed to be constant. Within a mode, an ordinal relationship exists between the 'groups' of alternatives. Within a group a cardinal relationship exists between the values of the alternatives. The top group, in a particular mode, may be defined to be 'mandatory'. Scheduling is based upon an admissions policy and a dispatching policy; both may make use of value and temporal attributes such as deadline. As well as being 'guaranteed' (or 'rejected') by the admissions policy, an alternative may be classified as 'rescindable' if it can be rejected at a later time. An alternative is 'completed' if it finishes before its deadline, otherwise it 'failed'.

3.4 Defining Modes in the Example

Clearly an autonomous vehicle has a number of operational modes depending upon the road and traffic conditions (environmental mode), the state of the on-board computing resources (computer mode) and the performance of related services (system mode). For illustrative purposes we choose just two major modes. The first, *day-time fault-free* (DFF) mode which implies a dry surface, good visibility and non-faulty hardware. The other mode *poor, wet and faulty* (PWF) concerns night time driving in wet conditions where some computing resources are malfunctioning. This represents a very extreme mode where no guarantees are possible and hence some form of best-effort scheduling is required.

In mode DFF two service groups are defined: a *mandatory* and an *optimal* group . The other mode, PWF, presents an example of graceful degradation; only one group is defined: *useful.* This mode is not mandatory as resources cannot be guaranteed.

The total set of services and alternatives for this example were given in Section 2. Table 2 presents the mapping of the services to the three groups. Note that *useful* is not the same as *mandatory*; *useful*, as well as not being mandatory, includes some of the services from *optional.* This not only reflects different environmental conditions but also the fact that even in mode PWF some value-added computations may be possible.

After value assignment has been addressed in Section 5 the example will be revisited.

4 Admission Control and Dispatching Using Values

In this section we use the framework to evaluate and compare published results in value-based scheduling. The available material is usefully partitioned into works that are concerned with scheduling systems with constant value (i.e., single moded within the framework) and those that address variable values. Static (off-line) approaches are considered first. We do not attempt to give a comprehensive review (as the size of the published material is too large), rather examples

SERVICE	DFF: Mandatory	DFF: Optional	PWF: Useful
Collision Avoidance	IRBD RADAR	SVT SRC	IRBD SVT RADAR SRC
Braking Control	Basic braking	ABS Load sensitive ABS	Basic braking ABS
Engine Control	Basic control	Increased precise(1) Increased precise(2)	Basic control Increased precise(1)
Lateral Control (input processing)	Magnetic markers	Line following	Magnetic markers Line following
Lateral Control	PID	FSLQ FRB NC	PID
Path Finding	Basic data fusion	Fuel optimisation Ride comfort Time optimisation	Basic data fusion Fuel optimisation
Route Planning		Middle level planning GPS global planning	Middle level planning
Displays Control	Basic update rates	Speed warnings Fuel consumption analysis	Basic update rates

Table 2: Autonomous Vehicle Example: Mode Characteristics

are given to illustrate the main approaches that have been developed. Moreover, it is not the intention of this paper to introduce any new value-based scheduling algorithms. We take the view that existing results (as referenced in this section) demonstrate the effectiveness and importance of value-based scheduling.

4.1 Off-line use of Value

The work of Bondavalli *et.al.* [5] uses value to statically determine the services that should execute in any particular mode to cope with run-time overload conditions. The system is assumed to execute in a reasonably limited number of modes and the attributes characterising the services are constant in any particular mode:

a) if the service execution violates its time constraints (including the case of non execution at all), a value l_i is associated to this event;

b) if a correct (both in time and value) outcome is produced, a value g_i is associated to this event;

c) if a timely but erroneous output is produced, a value p_i is associated to this event.

The three values (of which l_i and p_i have negative values) represent the different rewards the system obtains with the possible outcomes of a service S_i. Denoting with $\mathcal{P}_{n,i}$, $\mathcal{P}_{c,i}$ and $\mathcal{P}_{e,i}$ the probabilities of occurrence of the events described at points a), b) and c) respectively, the expected value V_i of an admitted instance of the service S_i can be obtained by the following weighted sum:

$$V_i \overset{def}{=} l_i\mathcal{P}_{n,i} + g_i\mathcal{P}_{c,i} + p_i\mathcal{P}_{e,i} \qquad (1)$$

In case the system resources are insufficient to execute the whole workload, it becomes necessary to select a subset of service requests to be discarded, that is to determine for each service S_i, $i = 1, 2, \ldots, n$, the percentage x_i of the instances selected for the execution. The following vector $\vec{x}$ is then derived which completely identifies the part of the workload admitted in the system:

$$\vec{x} = (x_1, x_2, \ldots, x_n)$$

In order to define a criterion which allows to choose among different such vectors, the "instantaneous" reward index $G(\vec{x})$ representing the expected total reward per unit of time obtained by using $\vec{x}$ is introduced:

$$G(\vec{x}) \overset{def}{=} \sum_{i=1}^{n} \frac{x_i}{T_i} V_i - \sum_{i=1}^{n} \frac{(1 - x_i)}{T_i} l_i \tag{2}$$

The definition of $G(\vec{x})$ exploits the additive property of the value function: the first summation refers to the expected value for the executed part of the workload, while the second one represents the total loss the system must pay for the non executed workload.

The optimal choice is then achieved (prior to execution) by selecting the vector $\vec{x}$ which leads to a maximisation of $G(\vec{x})$. Indicating with ρ^* the maximum utilization factor of the system resources determined by the scheduling algorithms employed locally to each processor, and with C_i and T_i the execution time and the period of each instance of the service S_i respectively, the following linear programming problem represents an adequate model for describing and solving such optimisation problem, where the maximised objective function (i) is exactly the instantaneous reward index $G(\vec{x})$:

$$\left\{ \begin{array}{lcc} \text{maximise} & \sum_{i=1}^{n} x_i(V_i/T_i) - \sum_{i=1}^{n}(1 - x_i)(l_i/T_i) & (i) \\ \text{subject to} & \sum_{i=1}^{n} x_i(C_i/T_i) \leq h\rho^* & (ii) \\ \text{and} & x_i \in [0, 1] & (iii) \end{array} \right. \tag{3}$$

The proposed linear programming problem is the continuous relaxation of the well-known "knapsack" problem [25] and can be solved very easily on the basis of value-density ordering. The idea is that all the services considered for execution are partitioned into classes according to a value-density criterion, which also assures a total ordering among the defined classes. Then, the selection of the services to execute starts from the class at the top of the ordering down to the less important ones until the executable workload is completed.

Zlokapa [38] develops off-line preprocessing algorithms that enable efficient and effective on-line scheduling of task groups with values, timing constraints, resource constraints, and precedence constraints. These algorithms derive new value densities that reflect how valuable the individual tasks and their successors are. By utilizing these reflective value densities, on-line schedulers are not required to examine the successors of the ready to execute tasks at run time to select the best task to schedule next. A performance study validates the utility of this approach.

4.2 Scheduling with Constant Values

A common assumption in the published literature is for value to be cardinal and constant for a particular service. Interestingly, although many papers have concerned themselves with value-based scheduling, few (if any) have directly addressed how the actual values are assigned. We shall return to this issue later.

With no admissions policy and pure value-based dispatching Locke [21] showed that executing services in decreasing order of value density (V/C) is optimal[1]. Here V is the constant value of the service (upon completion), and C is the worst case computation time of the service. By using value density, in essence, a cardinal function is assumed.

Locke also considered using value density to control admission, but earliest deadline scheduling of the actual processor. His *best-effort* scheme performed better (in simulation studies that ignored overhead cost) than any other simple scheme. However it does have complexity $O(n^2)$ in the number of active services. Buttazzo and Stankovic [7] improved upon the best-effort approach with their *RED* scheme. This has complexity $O(n)$ at each scheduling point.

Both best-effort and RED allow services to be rescinded, but there is an overhead cost in doing so. More recently Davis [8] investigated the use of a simple adaptive value-density threshold (AVDT) admission policy (with no rescinding). The system keeps a note of the current "Value-Density" performance of the system. A new service is immediately rejected if it does not meet this performance value. If it passes this hurdle a test is made to see if it can be admitted (i.e., will meet its deadline and all other committed services will still meet theirs). This test is made as new services arrive. At run-time fixed priority scheduling is employed. In simulation studies that do take account of overheads Davis showed that this AVDT scheme out performed other methods. This result is supported by Oliveira and Fraga [24] who considered interdependent services.

Buttazzo *et.al.* [6] through extensive simulation studies illustrated that during system underload earliest deadline scheduling produced best results; whilst during overload, value based (best effort) scheduling was best. This is a clear example of a mode change instigating different admission and dispatching policies.

A different form of analysis was used by Baruah *et.al.* [1] to show that the best an on-line algorithm can do (when compared with an optimal clairvoyant algorithm) is

$$\frac{1}{(1+\sqrt{k})^2}$$

where k is the ratio of the highest value service to the lowest value service.

Subsequently Koren and Shasha [18] introduced the D^{over} algorithm that was able to reach this upper bound (for a restricted model). The complexity of D^{over} is $O(\log n)$ at each scheduling point.

An example of a simple scheduling scheme is proposed by McElhone [23]. He has only three value-added classes. The admission policy within each class is FIFO. Scheduling is via fixed priority scheduling. The top class has a known worst case resources utilisation and is non-rescindable. The middle class also has known worst case resources usage but is weakly rescindable. The lower class is rescindable. In addition to these three groups there is a mandatory 'top' group. A service from this top group, although mandatory, may be replaced by a guaranteed non-rescindable service. This constitutes a minor mode change.

Value based real-time scheduling has been investigated in the real-time database (RTDB) area. For example, in [13, 11, 33, 12] transactions have both values and deadlines. Scheduling algorithms were developed and evaluated that choose priority by considering various combinations of value and deadline. Usually, total value is the performance metric and no value is accrued if deadlines are missed.

Also in the real-time database area, Bestavros and Braoudakis [2] examine the use of value in speculative concurrency control and admission control algorithms [3]. In both cases extensive simulation studies are performed.

[1]This observation is a generalisation of an earlier result by Smith [29].

All of these RTDB papers investigate the use of values via performance studies and clearly illustrate the benefits to be gained – they do not however describe how value is obtained in the first place.

4.3 Scheduling with Non-Constant Values

Where value for a service is allowed to change, the time-value scheme is one that has been considered in some detail. Here value is dependent on the completion time of the service (and may include negative value if a deadline is missed). Chen and Muhlethaler [16] produce an O(n^3) sub-optimal scheme for this model. Although this is considerable better than the O($n2^n$) optimal scheme, it is still too expensive to be used in reality.

Wedde *et.al.* [35] used value to priorities execution, if a task misses its deadline then its next release has an increased priority. This continues until either it meets its deadline or it is released with the highest priority (and then must meet its deadline). The algorithm is proposed for adaptive safety critical systems.

An alternative way in which values can change is through the allocation of more resources. In all the previous discussions it has been assumed that a service either completes by its deadline (in which case it has a value that is either constant or dependent on completion time) or it fails to finish in time. There are a class of routines, with are known in the AI community as 'anytime' algorithms [10], that increase their value as they are given more execution time. The notion of imprecise computation [28] also includes optional components with this property. Such routines are assumed to progress monotonically towards their optimal value. Performance profiles are used to approximate their progress. Scheduling with anytime algorithms presents particular difficulties, that are well reviewed by Zilberstein [37]; however there remains the problem of assigning the optimal values that each routine is ultimately capable of producing.

5 Assignment of Value

As indicated earlier there is actually very little literature in the scheduling field, on the actual assignment of value. This is somewhat surprising, without meaningful values the whole point of undertaking value-based scheduling is diminished. There are, however, a number of techniques that can be used to help produce sensible values. If value is deemed to be constant with a mode (**Definition 6**) then value assignment is an off-line activity.

Fortunately the disciplines of Measurement Theory [26] and Multiple-Criteria Decision Making [17, 34] offer mathematical frameworks and results for formulating and adequately approximating the value problem by a systematic analysis of the available decision alternatives and their consequences. There are two basic problems to be addressed in value assignment. First, knowing whether a value function *exists* that can 'represent' the decision maker's expressed preferences between the alternatives, and second, knowing how to construct such a value function in practice. These are known in the literature as the *representation* and the *construction* problem, respectively.

To understand the representation problem it is important to note that we cannot ask a general question such as "does a value function exist for this decision problem or not?"; the question has to be more specific, with respect to a particular numerical function: "Is it theoretically sound to use this specific functional form to represent the decision problem under consideration?". This question has been answered for a number of analytical forms of value functions (e.g., for a single attribute, the ordinal function V described in Section 3.1; for multiple attributes the additive, polynomial, geometric and probabilistic forms studied in [19, 31, 22] etc.).

Theoretical results are available that give sufficient conditions that must be fulfilled by binary preference relations (such as *More_Useful* defined in Section 3.1) for specific value functions to be used. If all the conditions in a collection of sufficient conditions are necessary as well, that is even better. A more important criterion is that the conditions be "testable" or empirically verifiable in some sense. The conditions are usually called *axioms* for the representation and the theorem stating their sufficiency is usually called the *representation theorem.*

The construction problem has been solved by various practical procedures that are based on the results of the representation theorems. One approach is to use pair-wise comparisons to produce a partial ordering and to use this to infer/invent a complete ordering. Pair-wise comparisons are intuitively simple and can be evaluated in a straightforward way. There are however other ways [17, 34] of obtaining sets of values. Many software tools are available that implement some of these procedures. For example, a tool called Expert Choice which is based on the Analytic Hierarchy Process [27] – a multiple attribute evaluation procedure, this tool is used in the case example is Section 6.

To give a concrete example of how representation axioms help us to assign values, consider the following representation theorem for a single attribute ordinal value function:

Theorem 5.1 *Suppose A is a finite set (with $a, b \in A$) and R is a binary relation on A. Then there is a real valued function V on A satisfying*
$aRb \Leftrightarrow V(a) > V(b)$
if, and only if, (A, R) is asymmetric and negatively transitive.

Thus, to see if we can measure preferences between service alternatives in a flexible real-time system to the extent of producing an ordinal value (utility) function, we simply check whether or not these preferences (as dictated by application specific requirements) satisfy the conditions of asymmetry and negative transitivity. In general, we could do this by doing a *pair comparison experiment.* For every pair of alternatives a and b in A, we (through a software tool) present a and b and ask the decision maker to tell us which if any they prefer. We present these pairs in a random order, and use their judgements to define preference. Using the data, we (or a tool) can check whether the axioms are satisfied; e.g., check whether they ever failed to prefer a to b, failed to prefer b to c, but said they preferred a to c.

There are two interpretations for the axioms. One is that these are testable conditions which describe what the empirical data must be like for measurement to take place. Thus the axioms could be used in a *descriptive* way, to simply ask whether or not the data satisfies these conditions. Alternatively, we could use these axioms to define *rationality.* We could then say that a decision maker who violates these axioms is acting irrationally. This approach is the *prescriptive* or *normative approach.* The representation theorem is used to define the class of attributes to which the theory applies — the so-called rational attributes. It is this second approach which we must apply if we use it to study measurement of say utility of service alternatives. For example, if R is the relation *More_Useful* we think of the conditions of asymmetry and negative transitivity as conditions of rationality, which must be satisfied before a measurement of (A, R) can take place in the number system $(\mathcal{R}, >)$.

It should be emphasised that an exact value function is not an absolute necessity. Rather the point of using tools such as Expert Choice is to minimise (rather than eradicate) internal inconsistencies. Preferences are the key notion in value based scheduling; the value function is just a useful way of representing these preferences.

5.1 Preference Modelling

So far, we have described a simple notion of preference which is only a part of a more comprehensive model for comparing alternatives. For any set A of entities, we can assume that a decision maker when asked to compare arbitrary elements a and b will react in one of the following three ways:

1. **Preference** for one of them (aPb or bPa)

2. **Indifference** between them (aIb)

3. **Incomparability**: refusal or inability to compare them (aJb)

We can assume [34] that the relations P, I and J will fulfil the following requirements:
$\forall a, b \in A:$
$aPb \Rightarrow \sim bPa : P$ is asymmetric
$aIa : I$ is reflexive
$aIb \Rightarrow bIa : I$ is symmetric
$\sim aJa : J$ is irreflexive
$aJb \Rightarrow bJa : J$ is symmetric

P, I and J are said to form a *preference structure* if they obey the essential axioms given above and if for all $a, b \in A$, one and only one of the following is true: aPb, bPa, aIb, aJb.

Note that these are only axioms that P, I and J are assumed to *always* obey. For a given set A and a specific attribute, they may obey other axioms depending on their contents as determined by the application. For instance, P may be transitive. For example consider the idea of service Classes given in Section 2: If a service in Class A is 'better' than one in Class B, and Class B is 'better' than Class C, then it follows that Class A is 'better' than C. However, I may be intransitive. For example, if service alternative a which is 'acceptably late' is deemed to be as good as service b which is 'optimally delivered', and in turn, b is deemed as good as c if c is 'acceptably early'. In this case, a few (or several) services down the line there will come a point when the accumulated time gains will result in a distinct preference for some service (say p) over service a. This is a demonstration of the intransitivity of I for the attribute time at which the service completes.

More sophisticated models of P, I and J may also be constructed, which take into account thresholds for preference. For example, the intransitivity of I can be removed by introducing a positive threshold q :

$$aPb \Leftrightarrow u(a) > u(b) + q$$

$$aIb \Leftrightarrow |u(a) - u(b)| \leq q$$

Thresholds can be used to evaluate questions such as how useful is it to increase the precision of some output by, say, 10%.

Gathering preference data is an intricate problem: experiments have shown that the way one puts questions to the decision maker may strongly influence their responses. Although there is no clear methodology to do this the exercise of trying to gather the data might in itself reveal useful information for measurement purposes. It could be used to confirm or dismiss the possibility of measurement (using a representation theorem) given the quality of available preference data.

5.2 Decomposition of Value Functions

Finding such a representation of value that is a function of n variables is a complex problem. Measurement theory and multiple attribute utility theory have attempted to decompose this large problem into smaller problems. For instance, it would be useful if we could break down a complex value function v into n simpler functions v_i which are each defined over a *single* attribute x_i. These individual value functions would then need to be combined together using some new function f. In other words,

$$v(x_1, x_2, \ldots, x_n) = f[v_1(x_1), v_2(x_2), \ldots, v_n(x_n)] \tag{4}$$

The hard work now gets shifted to finding an f that is suitable for the chosen $x_i s$ and $v_i s$. The literature contains various representation theorems giving necessary and sufficient conditions under which f has various analytical forms such as the additive function or a polynomial over the v_is. Hence, tool support can help the designer by taking them through a decision process to evaluate and compare different alternatives and services; thereby generating, if necessary, a cardinal value function.

5.3 Evaluating an Inconsistency Index for Pair-Wise Entries

One example of a tool supporting value assignment is Expert Choice (EC). It allows relevant factors to be organised into a hierarchy (goals, criteria, sub-criteria etc) and supports relative and absolute measurement. Although for some attributes absolute measurements may be available, in general an approach based upon relative measurements is more appropriate. Given a set of services, EC supports pair-wise comparisons and builds up an internal representation of rankings and preferences.

If we have n service alternatives $A_1, \ldots A_n$, then we can specify the decision maker's pairwise preferences between them using a matrix P as shown below. The (i, j)th element of P, p_{ij} represents the relative importance of alternative A_i with respect to A_j. This naturally leads to a *positive reciprocal matrix* (where any $p_{ij} = 1/p_{ji}$ and whose leading diagonal elements are 1).

$$P = \begin{bmatrix} 1 & p_{12} & \ldots & p_{1n} \\ 1/p_{12} & 1 & \ldots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 1/p_{1n} & 1/p_{2n} & \ldots & 1 \end{bmatrix}$$

If there are n service alternatives, then at least $(n-1)$ comparisons must be made and a total of $n(n-1)/2$ are needed to give complete coverage. Various graphical and tabular user-interface methods are supported by EC to enable these pair-wise comparisons to be obtained. It must be emphasised that extracting pair-wise values does not guarantee that the real preferences of the engineers are been articulated. Nor does it ensure that the real utility of a service is exactly correlated to the value assigned to it. Nevertheless, it is possible to check for internal inconsistencies in the value entry process.

It turns out that the consistency of a positive reciprocal matrix is equivalent to the requirement that its maximum eigenvalue λ_{max} should be equal to n [27]. It is also possible to estimate the departure from consistency by the difference $\lambda_{max} - n$ divided by $n - 1$. This is known as the *consistency index* (CI). We note that $\lambda_{max} \geq n$ is always true. How bad our consistency may be in a given problem may be estimated by comparing the calculated value of CI with the value obtained from randomly chosen judgements in a matrix of the same size. When applied to the randomly chosen judgements CI is termed the *random index* (RI).

A table of the average RI for matrices of order 1 to 15 has been generated by laboratory experiments. The ratio of CI to RI for the same order matrix is the *inconsistency index* (or ratio). An inconsistency index of 0.10 or less is considered acceptable [27]. With such a low index it is assured that there are no ordinal inconsistencies, but there many be some residual cardinal ones (e.g. A is twice the value of B and B is twice the value of C, but A is only 3.95 times the value of C).

Following data entry, EC produces two outputs: a string of real numbers (between 0 and 1) that represents the best fit of values to services; and the inconsistency index.

5.4 Summary

The assignment of value, which is essential to most forms of best-effort scheduling, can never be accomplished in a totally deterministic manner. It will inevitably entail the use of intuition and judgement. Nevertheless, measurement theory and multiple-criteria decision making provide frameworks that allow a systematic, and less error prone, approach to value assignment to be achieved.

An effective approach is to use pair-wise comparisons between alternatives. And to do this on an attribute by attribute basis. Having derived these comparisons, the calculation of the inconsistency index allows the quality of comparisons to be evaluated. Tool support will enable the actual values to be obtained (that is, the values that minimise the inconsistency index).

Once modes and values have been derived they can be combined with a value-based scheduling scheme to furnish an effective run-time decision process.

6 An Example of Value Assignment

We shall look at just one mode of operation: DFF. In this mode there were two groups of services. The first group is however mandatory, so value-based scheduling is only applicable to the second (optional) group. As shown in Table 2 this group consists of 8 services and 17 service alternatives. Our objective is to obtain a value for each alternative that can be used for best effort scheduling during mode DFF. We shall assume that a domain expert can compare the alternatives using a single attribute.

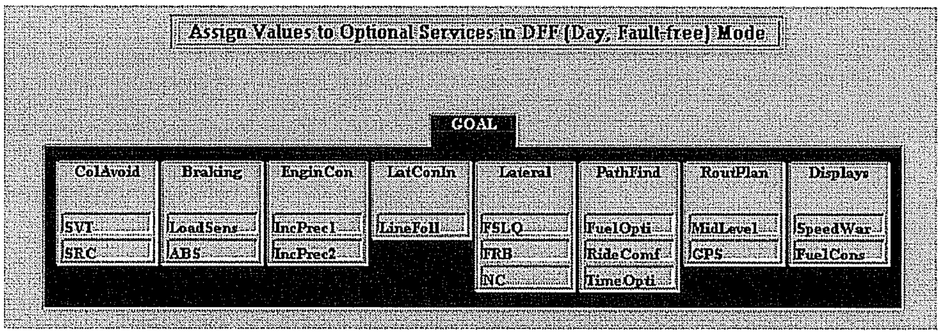


Figure 1: The Decision Model for Optional Services in the DFF Mode

The decision model constructed for the problem is shown in Figure 1. It is a direct mapping of Table 2 into an Expert Choice hierarchy. The abbreviations used in the model are shown in Table 3. Note the services become "goals" and the alternatives means of achieving that goal.

Abbreviation	Definition
ABS	Anti Lock Braking
Braking	Braking Control Service
ColAvoid	Collision Avoidance Service
Displays	Displays Control Service
EnginCon	Engine Control Service
FRB	Fuzzy Rule based
FSLQ	Frequency-Shape Linear Quadratic
FuelCons	Fuel Consumption analysis
FuelOpti	Fuel Optimisation
GPS	GPS global planning
IncPrec1	Increased Precise 1
IncPrec2	Increased Precise 2
LatConIn	Lateral Control (input processing)
Lateral	Lateral Control Service
LineFoll	Line Following
LoadSens	Load sensitive ABS
MidLevel	Middle level planning
NC	Neuro Control
PathFind	Path Finding Service
RideComf	Ride Comfort
RoutPlan	Route Planning Service
SRC	Short Range Communication
SVT	Stereo Vision based Vehicle Tracking
SpeedWar	Speed Warnings
TimeOpti	Time Optimisation

Table 3: Abbreviations used in the DFF Mode Decision Model

6.1 Comparison Procedure

A full analysis of 17 alternatives would require $(17 \times 16)/2 = 136$ pair-wise comparisons (judgements). This is too large to do in practice. Hence, we use a different scheme that allows us to reduce the number of comparisons. The procedure we use has the following three steps:

1. *Full comparison of alternatives within each service.* For each service, consisting of n alternatives, we need ${}^{n}C_2$ comparisons. Thus for the 8 services shown in the figure, we require $1+1+1+0+3+3+1+1 = 11$ comparisons.

2. *Minimal number of comparisons between the 'best' alternatives of each service.* Since there are 8 services, this means $8 - 1 = 7$ judgements. This should result in an initial ranking of all the 17 alternatives using a total of 18 pairwise comparison judgements (11 from the intra-service comparisons, and 7 inter-service ones).

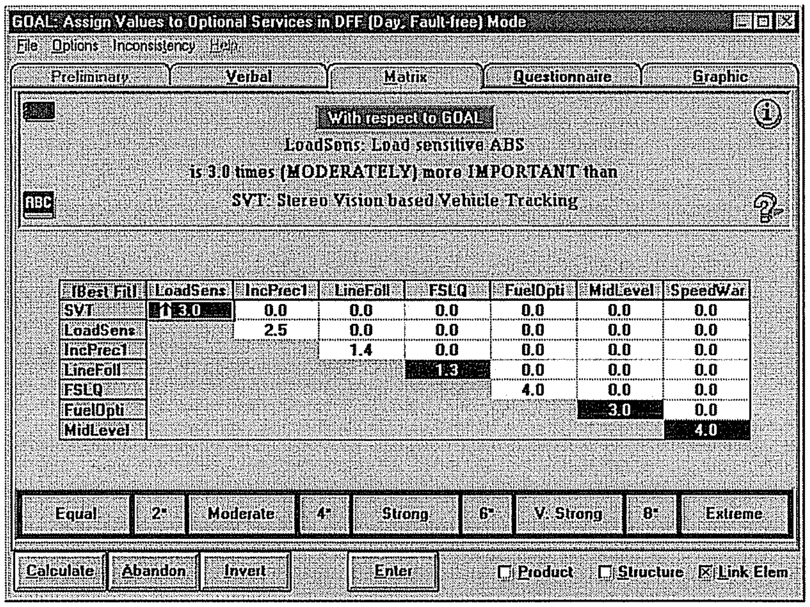
3. *Full comparison between the best alternatives of each service.* For the 8 services this means $(8 \times 7)/2 = 28$ judgements altogether, and hence 21 more than the minimal. The total number of comparisons is thus $11 + 28 = 39$. The final ranking may require more iterations on the judgements due to the inconsistency index check. Nevertheless the total number of comparisons needed is now $11 + 28 = 39$.

The 'Link Elements' command in Expert Choice is a means of carrying out the procedure exactly as described above. Linking nodes relies on the idea that if $a = b$ and $c = d$, then the relationship between b and c also gives the relationship between a and d. Thus, the 'Link Elements' command is based on the assumption that if your judgement about the relation between a and b

which are both under one node, is accurate and consistent, and that if your judgement about the relationship between c and d which are under another node is accurate and consistent, then an accurate judgement about the relationship between b and c also yields the relationship between a and d.

This method of linking groups of nodes in the hierarchy uses the first elements from the respective groups in the linking comparisons. For this reason, while constructing the model in Figure 1, we ensure that the 'best' alternative is the first one to be inserted under each service.

6.2 Observations and Results



With respect to GOAL
LoadSens: Load sensitive ABS
is 3.0 times (MODERATELY) more IMPORTANT than
SVT: Stereo Vision based Vehicle Tracking

(Best Fit)	LoadSens	IncPrec1	LineFoll	FSLQ	FuelOpti	MidLevel	SpeedWar
SVT	↑ 3.0	0.0	0.0	0.0	0.0	0.0	0.0
LoadSens		2.5	0.0	0.0	0.0	0.0	0.0
IncPrec1			1.4	0.0	0.0	0.0	0.0
LineFoll				1.3	0.0	0.0	0.0
FSLQ					4.0	0.0	0.0
FuelOpti						3.0	0.0
MidLevel							4.0

Figure 2: Minimal Set of Pairwise Comparisons using Linking Elements

We present here the outcome of the Value assignment experiment. Step 1 of the procedure described in the previous section is easy to carry out provided there is domain knowledge and engineering data available to support the comparisons between the alternatives within each service. For the autonomous vehicle example, we believe this to be the case.

The action of Step 2 is shown in Figure 2. The 'best' alternative for each service is represented as the rows and columns of a *comparison matrix*. The tool user (domain expert) fills in the diagonal entries and therefore provides the minimum set of comparisons. Various options in the tool allow different forms of data entry but they all result in entries in the comparison matrix.

Having made these entries, the tool will calculate the inconsistency index and the actual real numbers that are the alternative's values. Because Step 2 only involves a minimal set of comparisons the inconsistency index is zero (that is, fully consistent). Figure 3, from the tool, provides this data and a read out for the values.

The action of Step 3 is given in Figure 4. Now all the entries are filled in. Note only half the matrix is used as the preference of A over B is used to directly calculate the B over A preference (it is the inverse). Output from Step 3 is given in Figure 5. The discrepancy between the two sets of results is noteworthy. Once Step 3 has been undertaken inconsistencies arise. But an inconsistency index of 0.09 is acceptably small.

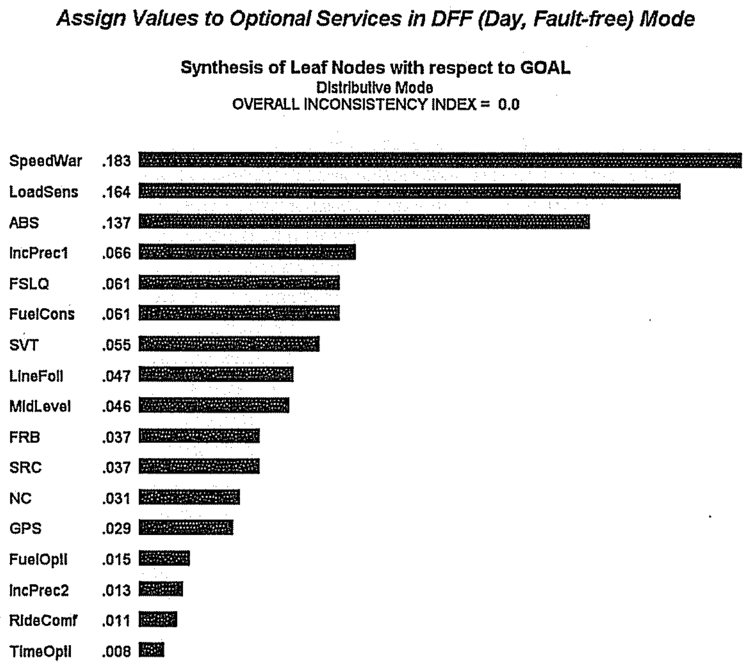
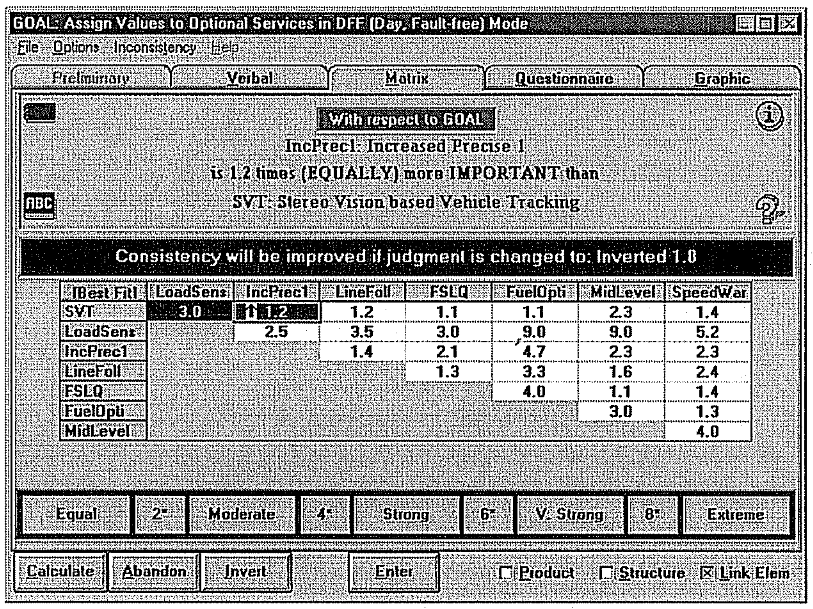


Figure 3: Values Derived from Minimal Judgements, with Zero Inconsistency Index



With respect to GOAL

IncPrec1: Increased Precise 1
is 1.2 times (EQUALLY) more IMPORTANT than
SVT: Stereo Vision based Vehicle Tracking

Consistency will be improved if judgment is changed to: Inverted 1.0

[Best Fit]	LoadSens	IncPrec1	LineFoll	FSLQ	FuelOpti	MidLevel	SpeedWar
SVT	3.0	↑ 1.2	1.2	1.1	1.1	2.3	1.4
LoadSens		2.5	3.5	3.0	9.0	9.0	5.2
IncPrec1			1.4	2.1	4.7	2.3	2.3
LineFoll				1.3	3.3	1.6	2.4
FSLQ					4.0	1.1	1.4
FuelOpti						3.0	1.3
MidLevel							4.0

☐ Product ☐ Structure [✓ Link Elem]

Figure 4: Full Set of Pairwise Comparisons using Linking Elements

The output of the value assignment process (as shown in Figure 5) is a set of values for the mode DFF. Once scaled to integers, they range from 219 for Load Sensitive ABS, to 14 for Fuel Consumption Analysis. These values and other characteristics such as worst case computation times can then be used by the run-time scheduler.

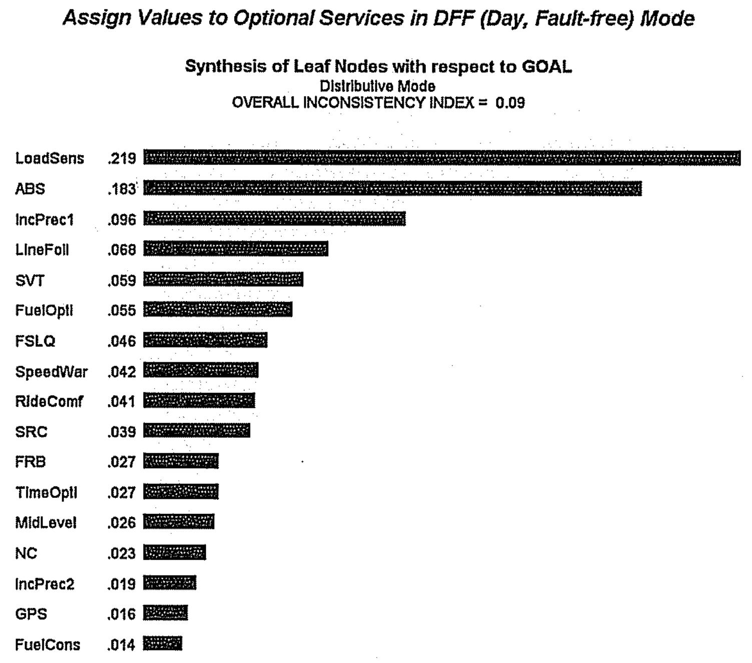


Figure 5: Values Derived from Full Comparisons, with Non-Zero Inconsistency Index

7 Conclusion

Although there is considerable literature in the real-time community on flexible scheduling and adaptive systems, this work lacks an engineering context. This paper has attempted to formulate a framework for value-based scheduling that will enable this technique to be used in practice. It has shown that general models must take into account run-time overheads if effective dynamic behaviour is to be realised.

It is surprising that there is actually very little literature in the scheduling field, on the actual assignment of value. An approach is developed in this paper that allows values to be assigned off-line (on a per mode basis and assuming constant value within a mode). To cater for the requirement that some services have value incomparable with other services, the alternatives available in a particular mode are split into groups. An ordinal relationship exists between the groups. Within a group a cardinal value function is deemed to exist. At run-time changes between modes can accommodate various forms of flexible scheduling and can approximate time-varying value functions.

To derive a cardinal value function (i.e., set of values) within a particular mode and group, it is recommended that a technique based on pair-wise comparisons is employed. Assessment routines are available that allow the consistency of a set of pair-wise comparisons to be evaluated and constant values to be generated that furnish maximum consistency. These values can then be used by a range of flexible scheduling techniques that are described in the literature.

Future work will attempt to include notions of confidence into the pair-wise comparisons (that is, make use of the knowledge that some judgements are based on sounder engineering data). It will also be necessary to more rigorously integrate multiple attribute functions into the pair-wise comparisons technique. It remains a open question whether time-varying value functions can be adequately approximated by this (or any other) approach.

Ultimately the evaluation of the approach supported in this paper will come from application experience in an appropriate industrial context. The case study described in this paper will be expanded with the intention of including more domain knowledge and practical experience. It is intended that this will form the basis for an extensive evaluation of the approach.

References

[1] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the competitiveness of online real-time task scheduling. *Real-Time Systems*, 4(2):124–144, 1992.

[2] A. Bestavros and S. Braoudakis. Value-cognizant speculative concurrency control. In *Proceedings VLDB 95, Zurch, Switzerland*, 1995.

[3] A. Bestavros and S. Nagy. Value-cognizant control for rtdb systems. In *Proceedings IEEE Real-Time Systems Symposium*, pages 203–239, 1996.

[4] S. Biyabani, J.A. Stankovic, and K. Ramamritham. The integration of criticalness and deadlines in scheduling real-time tasks. In *Proceedings of the 9th IEEE Real-Time Systems Symposium*, pages 152–169, 1988.

[5] A. Bondavalli, F. Di Giandomenico, and I. Mura. Value-based resource assignment in object-oriented real-time dependable systems. In *Proceedings WORDS'97*, 1997.

[6] G. Buttazzo, M. Spuri, and F. Sensini. Value vs. deadline scheduling in overload conditions. In *Proceedings of the IEEE Real-Time Systems Symposium*, 1995.

[7] G. Buttazzo and J. Stankovic. RED : A robust earliest deadline scheduling algorithm. In *Proceedings of the 3rd International Workshop on Responsive Computer Systems, Austin*, 1993.

[8] R. I. Davis, S. Punnekkat, N. C. Audsley, and A. Burns. Flexible scheduling for adaptable real-time systems. In *Proceedings IEEE Real-Time Technology and Applications Symposium*, Chicago, USA, 1995.

[9] R.I. Davis, K.W. Tindell, and A. Burns. Scheduling slack time in fixed priority pre-emptive systems. In *Proceedings Real-Time Systems Symposium*, 1993.

[10] T.L. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, 1988.

[11] J. Haritsa, M. Carey, and M. Livny. Value-based scheduling in real-time database systems. Technical Report TR 1204, Dept. CS, Univ. of Wisconsin, Madison, 1991.

[12] J. Haritsa, M. Livny, and M. Carey. Earliest deadline scheduling for real-time database systems. In *Proceedings 12th IEEE Real-Time Systems Symposium*, 1991.

[13] J. Huang, J. Stankovic, D. Towsley, and K. Ramamritham. Experimental evaluation of real-time transaction processing. In *Proceedings of the 10th IEEE Real-Time Systems Symposium*, 1989.

[14] IEEE Industrial Electronics Society. *Proceedings of the Intelligent Vehicles '92 Symposium*, June 1992.

[15] E.D. Jensen, C.D. Locke, and H. Tokuda. A time driven scheduling model for real-time operating systems. In *Proceedings IEEE Real-Time Sytems Symposium*, pages 112–122, 1985.

[16] Chen K and P. Muhlethaler. A scheduling algorithm for tasks described by time value function. *Real-Time Systems*, 10(3):293–312, 1996.

[17] R.L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs.* John Wiley & Sons, 1976.

[18] G. Koren and D. Shasha. *d^over*: An optimal online scheduling algorithm for overloaded real-time systems. In *Proceedings IEEE Real-Time Systems Symposium*, pages 290–299, Raleigh-Durham, North Carolina, 1993.

[19] D.H. Krantz, R.D. Luce, P. Suppes, and A. Tversky. *Foundations of Measurement: Additive and Polynomial Representations*, volume 1. Academic Press, 1971.

[20] J.P. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks fixed-priority preemptive systems. In *Proceedings 13th Real-Time Systems Symposium*, pages 110–123, 1992.

[21] C.D. Locke. Best-effort decision making for real-time scheduling. CMU-CS-86-134 (PhD Thesis), Computer Science Department, CMU, 1986.

[22] R.D. Luce, D.H. Krantz, P. Suppes, and A. Tversky. *Foundations of Measurement: Representation, Axiomatization, and Invariance*, volume 3. Academic Press, 1990.

[23] C. McElhone. *Flexible Scheduling in Hard Real-Time Systems.* PhD thesis, Department of Computer Science, University of York, UK., March 1994.

[24] R. Oliveira and J. Fraga. Scheduling imprecise computation tasks with intra-task/ inter-task dependence. In *Proceedings of the 21st IFAC/IFIP Workshop on Real-Time Programming, WRTP'96*, pages 51–56, 1996.

[25] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., 1982. ISBN 0-13-152462-3.

[26] F.S. Roberts. *Measurement theory, with applications to decision-making, utility and the social sciences.* Addison Wesley, 1979.

[27] T.L. Saaty. *Multiple Criteria Decision Making: The Analytic Hierarchy Process.* RWS Publications, 1992.

[28] W. K. Shih, J. W. S. Liu, and J. Y. Chung. Algorithms for scheduling imprecise computations with timing constraints. In *Proc. IEEE Real-Time Systems Symposium*, 1989.

[29] W. E. Smith. Various optimisers for single-stage production. *Naval research and Logistics Quarterly*, 3(1), 1956.

[30] J.A. Stankovic and K. Ramamritham. The spring kernel: A new paradigm for real-time operating systems. *ACM Operating System Review*, 23(3):54–71, 1989.

[31] P. Suppes, D.H. Krantz, R.D. Luce, and A. Tversky. *Foundations of Measurement: Geometrical, Threshold and Probabilistic Representations*, volume 2. Academic Press, 1989.

[32] H. Tokuda, J.W. Wendorf, and H.Y. Wang. Implementation of a time-driven scheduler for real-time operating systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 271–280, 1987.

[33] S. Tseng, Y. Chin, and W. Yang. Scheduling real-time transactions with dynamic values: A performance evaluation. In *Proceedings 2nd Int IEEE Workshop on Real-Time Computing Systems and Applications*, 1995.

[34] P. Vincke. *Multicriteria Decision-Aid.* John Wiley and Sons, 1992.

[35] H. Wedde, K. Stange, and J. Lind. Integration of adaptive file assignment into distributed safety-critical systems. In *Proceedings of the 21st IFAC/IFIP Workshop on Real-Time Programming, WRTP'96*, pages 37–44, 1996.

[36] J.W. Wendorf. Implementation and evaluation of a time-driven scheduling processor. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 172–180, 1988.

[37] S. Zilberstein. Resource-bounded sensing and planning in autonomous systems. *Autonomous Robots*, 3:31–48, 1996.

[38] G. Zlokapa. *Real-Time Systems: Well-Timed Scheduling and Scheduling with precedence constraints.* Ph.D. thesis, Department of Computer Science, University of Massachusetts at Amherst, 1993.