ABSTRACT

Title of Thesis:	EXPLOITING SOFT COMPUTING FOR REAL TIME PERFORMANCE
	Priyanka Rajkhowa, Master of Science, 2006
Thesis directed by:	Dr. Donald Yeung, Department of Electrical and Computer Engineering

The classic approach to the design of real time systems is to determine worst-case scenarios for the system statically and manually and then build the system with sufficient resources to meet deadlines and goals. This approach has worked well for traditional real time systems which operate in relatively simple, well-characterized environments. However the emerging generation of complex, dynamic and uncertain real time application domains accentuates the growing need for flexible, adaptable design approaches for real time systems. With the increasing complexity of real time systems, it is becoming infeasible to build systems with sufficient resources to meet the functional and timing requirements of all application tasks at all times. What is becoming increasingly important in the new paradigm of real time computing is the need to meet deadlines with sufficient system solution quality without having to design the system to support worst case program execution. In this thesis, we explore the possibility of exploiting "soft computing" properties of kernels to meet this objective. The chief characteristic of "soft computations" is the fact that they are able

to provide cruder results before they complete, or they may execute for a long time refining an already adequate result. In other words, such computations are able to provide useful/incremental results before fully completing execution. More specifically they provide a trade-off between computation time and algorithm solution quality.

This thesis addresses the design issues involved in building a system that exploits the "soft computing" properties of kernels to optimize real time performance. In this context, we make the following contributions. Firstly we build a system prototype of a real time situational assessment scenario. We thereafter identify "soft computations" in the system and characterize the computation time/solution quality trade-off opportunities provided by them using performance profiles. Thirdly, we introduce a method to use performance profile based models at run time to determine the optimal composition of different "soft computations" in order to meet real time deadlines with sufficient system solution quality. We quantify the gains from our method both in terms of functional correctness of the system as well as CPU utilization as compared to conventional real time scheduling techniques. We observe that our dynamic scheduling scheme on an average is able to meet the system goals with 39% more accuracy with no missed deadlines as compared to conventional real time scheduling techniques for various design points that do not support worst case behavior. In addition, our method is able to meet the system objective while being highly utilized. Most importantly, our scheme exploits the soft computing properties of kernels to facilitate the design of the system at less aggressive design points while meeting deadlines and system goals at the same level as conventional real time design

methodology. Finally, we perform an experimental study to understand the sensitivity of performance profiles to various input data parameters and identify the potential for online learning of performance profiles.

EXPLOITING SOFT COMPUTING FOR REAL TIME PERFORMANCE

By

Priyanka Rajkhowa

Thesis submitted to the Faculty of the Graduate School of the

University of Maryland, College Park, in partial fulfillment

of the requirements for the degree of

Master of Science

2006

Advisory Committee:

Dr. Donald Yeung (Chair and Advisor)

Dr. Bruce Jacob

Dr. Manoj Franklin

© Copyright by
Priyanka Rajkhowa
2006

Acknowledgements

The list of people who have played a part in enriching my research experience and making this thesis a reality is long and I really hope that I do not leave anyone out. First and foremost, I would like to thank my research advisor, Dr. Donald Yeung for steering me in the direction of my topic and also providing endless guidance and support to an inexperienced researcher like me throughout the research process. This thesis would not have been possible without his valuable advice and the countless hours he has spent discussing the issues and problems I have faced. I sincerely appreciate his valuable contributions to my enriching experience as a graduate student.

I would also like to extend my gratitude to Dr. Bruce Jacob and Dr. Manoj Franklin for agreeing to be part of my Thesis committee.

I thank my research group members and the members of the Computer Architecture laboratory who have served both as my colleagues in research and as my friends. In addition, I would like to acknowledge and thank all the members of the CEARCH team at the Information Sciences Institute, Stanford University and Northrop Grumman for their help and support in the course of my thesis project work. Without Northrop Grumman's simulation framework, the work done as a part of this thesis might not have been achievable in such a short duration.

I thank my husband, Dipanjan, for his constant love, support, patience and encouragement and for being by my side at all times. I am indebted to my parents and brother for their emotional support and prayers. Without their love and prayers, the work done as a part of this thesis would not have been possible.

A very special note of thanks to my friends Bhuvan, Smitha, Amit, Alokika, Ashish and Sebastian. Their love, help and support has really enriched my graduate school experience.

Over and above everything else, I am really grateful to God for blessing me with a number of opportunities, giving me the strength to reach this point, and for providing me with great family and friends.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Motivation

Real time computing systems are defined as those systems in which the correctness of the system depends not only on the logical results of the computation, but also on the time at which they are produced [53]. The objective of real time computing is to meet the timing and functional requirements of individual tasks. Additionally it is also desirable that real-time systems achieve their functional correctness and timeliness while being highly utilized.

Real time systems are specified by a set of timing constraints, called deadlines. The objective of the system is to provide system solutions with requisite functional correctness by the specified deadlines. Based on the nature of the timing constraints that they need to satisfy, real-time applications and systems can be characterized as "hard" real time systems or "soft" real time systems. In a hard real-time system, if one or more activities miss a deadline or timing constraint, the system fails. In contrast, a soft real-time system is one that does have timing requirements, but occasionally missing them has negligible effects, as application requirements as a whole continue to be met.

When activities have timing constraints, as is typical of real time computing systems, scheduling these activities to meet their timing constraints is one major design issue that needs to be addressed. Traditional real time scheduling algorithms fall into two categories: static and dynamic. A static/pre-runtime approach calculates (or predetermines) schedules for the system off-line. It requires prior knowledge of all task characteristics (arrival times, deadlines, release times and worst case execution times). In a dynamic or runtime approach, the order of execution of tasks is decided at runtime based on priorities attached to tasks. These priorities in turn may be determined off-line (like in the case of the Rate Monotonic Algorithm[1] where priorities are based on the frequency/periods of tasks) or online (like in the case of the Earliest Deadline First algorithm (EDF) [1], where a task with the earliest deadline is given highest priority). These scheduling techniques will be discussed in detail later in Chapter 2.

1.1 Real time systems: The current paradigm

Traditionally the real time computing paradigm has included applications from the arena of digital control, digital signal processing, multimedia and database transactions [53].

Traditional real time systems like digital controllers operate in relatively simple, well-characterized environments. Such "traditional" real-time systems have a set of repeated tasks with known execution times and arrival patterns. The primary challenge in building such systems is to schedule these independent periodic tasks and ensure that they meet their deadlines. Classic real-time system approach is to determine worst-case scenarios statically and manually, then build systems with sufficient resources to meet goals.

This approach works well only under the following scenarios:

- it is possible to develop an accurate workload model of the environment in which the real time system operates. This ensures that Worst Case Execution Time

(WCET) estimates for task execution times, arrival and release times are accurate and hence guarantees that all timing constraints of tasks are met.

- the variance of actual execution times of tasks with reference to the WCET estimates is low. This ensures that in addition to meeting the timing and functional constraints of the system, the real time system is not underutilized.

However many complex real world applications such as real time database systems, agile manufacturing, robotics and various command and control systems work in unpredictable dynamic environments [3, 4, 11, 23, 26]. Accurate knowledge about the resource and data requirements of many of these tasks is not known apriori because the execution time and resource requirements of many of these tasks may be dependent on input data (information and decision support system) or dependent on sensor values (manufacturing plant, command and control system). The application of dynamic priority driven scheduling algorithms like Earliest Deadline First (EDF) [1] have been shown to be amenable to be applied to such systems. However due to the unpredictably of dynamic schemes under overload conditions, current commercial real time applications executing in dynamic environments use static priority driven algorithms like Rate Monotonic (RMA) [1] and hence are generally over designed /underutilized.

1.2 Real time computing: The next generation

The next generation of real time systems will have much more complex system requirements and will work in uncertain dynamic environments [26, 27, 38, 39, 40, 41]. Hence the need for adaptable, flexible behavior in such systems would be of primary importance. This need is accentuated by the following emerging trends in this field:

- Convergence of two major areas in computer science and engineering: Artificial Intelligence (AI) and real time systems. AI systems are moving towards more realistic domains requiring real time responses and real time systems are moving towards more complex applications requiring intelligent behavior [39, 41].
- The growing need for real time fusion of huge amounts of data into synthesized information in domains like avionics, medicine, defense, integrated vision, robotics, finance/business etc.[26, 41, 44, 45, 46, 48] to facilitate real time decision support and diagnosis.

Figure 1 illustrates the general structure of an emerging/next generation real time computing system. In general, such a system would comprise a set of objects in a scenario being monitored and controlled in real time by a computing system. The computing system would be periodically provided with raw data about various aspects of the objects and the scenario; the objective of the system would be to process the plethora of data being supplied by various monitors and sensors, correlate the various forms of data into information and use this information to update plans/ diagnosis etc. In addition, the system may be required to provide the information so synthesized to the human operators the system interfaces with and also control various aspects of the objects in the world to improve the state of the system.

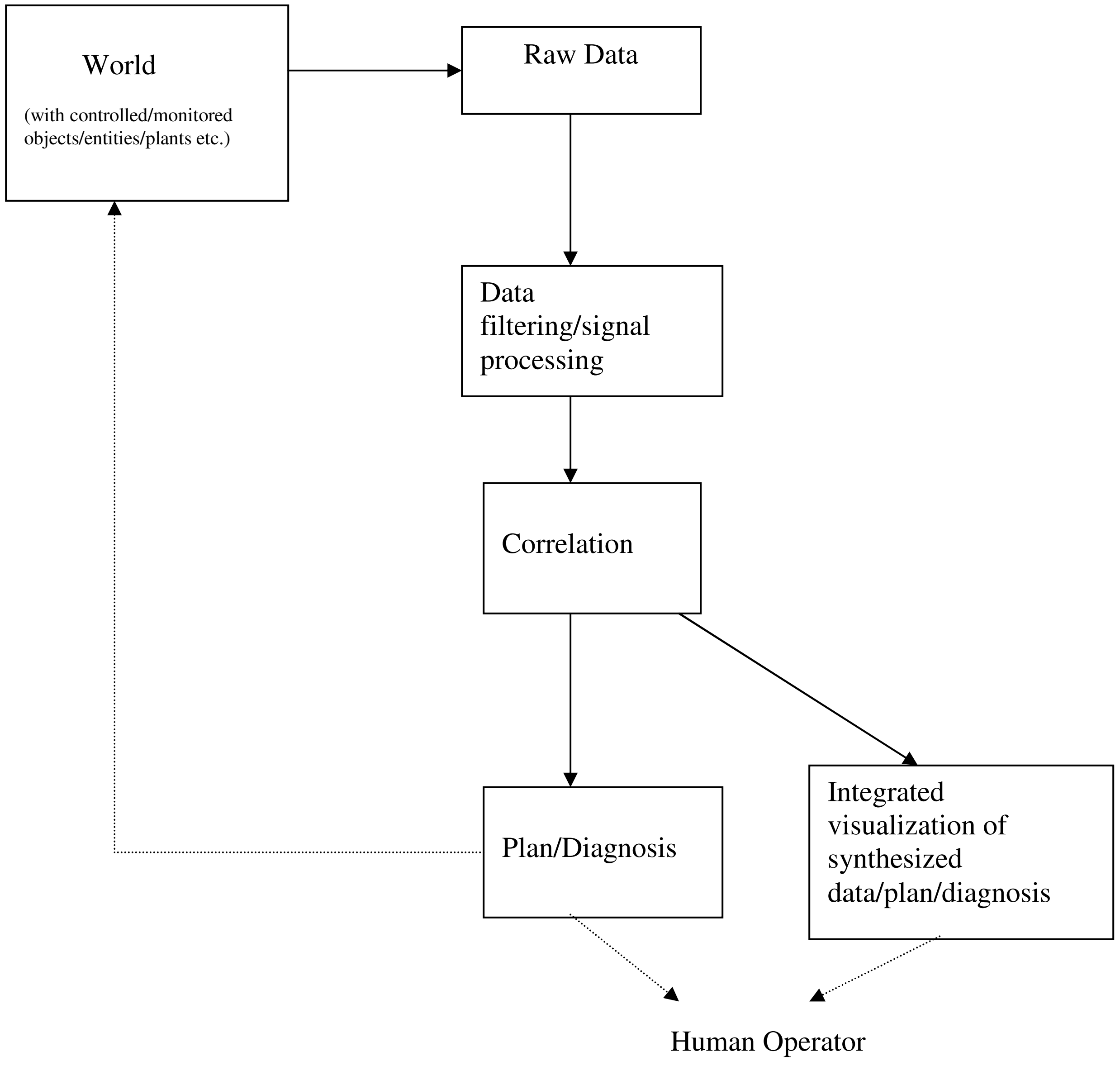

World
(with controlled/monitored objects/entities/plants etc.)
Raw Data
Data filtering/signal processing
Correlation
Plan/Diagnosis
Integrated visualization of synthesized data/plan/diagnosis
Human Operator

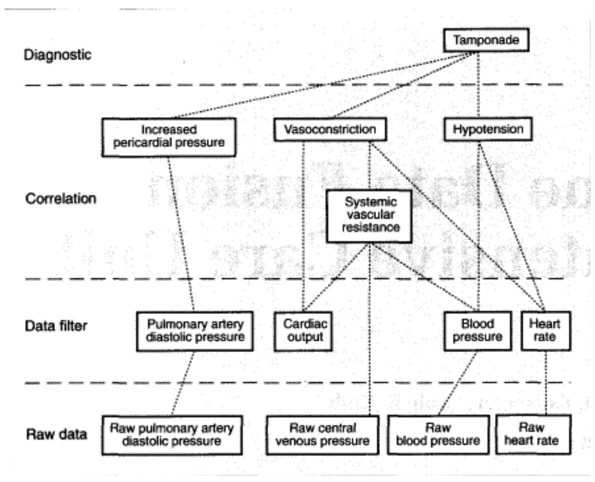
Figure 1: General structure of the next generation real time computing system

In an open loop system (where the system only interfaces with human operators), the time constraint would be to keep up with the data produced by the world and provide synthesized information in real time to the human operators. In the closed loop system (where the system feedbacks synthesized information to the world), the real time constraint would include the latency to control the objects in the world (wherever applicable) so as to ensure temporal consistency of actions taken based on processed data. If we look at the system illustrated in Figure 1, it comprises applications pertaining to different domains including digital control, signal processing, multimedia and artificial intelligence. Systems like this have existed for a long time in the medical domain, defense and air traffic command and control systems, diagnosis and trouble shooting of devices of all kinds, manufacturing process planning, job-shop scheduling; the difference between the currently existing systems and systems that researchers envision for the future is the levels at which the system interfaces with human operators and the monitored/controlled world and hence the demarcation of work between the machine and the human operators. The scenario we discuss below will make this assertion more clear.

Let us consider the example of real time medical data fusion in the intensive care unit [41, 45] to understand the paradigm changes in the real time computing arena. In today’s Intensive Care Unit (ICU), patients are surrounded by a battery of instruments. Each performs a different monitoring task, generating masses of low-level summary data and each device connected to the patient is separately controlled. The clinician must integrate the data they generate, decide what is important and what is irrelevant, and synthesize a high-level overview. If the patient goes into some type of trauma (like shock), alarms sound to summon medical personnel. During the time it takes for them to arrive, the patient’s condition deteriorates. Even worse, upon entering the room, medical personnel must waste precious time trying to figure out what has happened, why, and what to do about it. Faced with overwhelming data, the clinician may focus on a subset of the signals, ignoring other significant information and overlooking problems. A computer system, that could see the whole picture (integrating the results of many sensor readings), could initiate small actions (such as adjusting the feed rate on a respirator) that might prevent the patient from experiencing the trauma. In addition, if the patient did go into shock, the system could diagnose the cause and have diagnostic (and treatment) suggestions ready by the time medical personnel arrived. In cases where immediate steps must be taken, the system could initiate precursor actions such as reducing a particular gas in a ventilator, which should be done if emergency surgery may be required. Thus a system that integrates data from multiple sources could present a high-level synthesis, minimizing information overload and preventing fixation.

To build such a real time system, we must achieve predictable, real-time performance, accommodate heterogeneous approaches to the many separable sub problems, and design a useful interface. Substantial computing power is needed to process many continuous waveforms, convert them into a qualitative representation, correlate information to identify the intermediate-level physiological state, and finally produce high-level diagnoses and suggestions.

Figure 2a illustrates the components of the real time system we discussed above and also shows how each of the computing components pertaining to this system in the medical domain fits into the general structure of the next generation real time computing system we illustrated in Figure 1. Figure 2b shows the kind of specifications a real time computing system in the ICU, as described above, is likely to have. Firstly such a system would have a hybrid system model pertaining to a mixture of time triggered (periodic) and event triggered (aperiodic tasks). In addition, the timing constraints of such a system would be hybrid, meaning the system would have to meet a mixture of hard and soft deadlines.



Diagnostic
Tamponade
Increased pericardial pressure
Vasoconstriction
Hypotension
Correlation
Systemic vascular resistance
Data filter
Pulmonary artery diastolic pressure
Cardiac output
Blood pressure
Heart rate
Raw data
Raw pulmonary artery diastolic pressure
Raw central venous pressure
Raw blood pressure
Raw heart rate

System Model:

- Hybrid model :Time and event triggered model

Timing requirements:

- Hybrid: Both soft and hard/deadlines

Figure 2a: Components of a real time data fusion system in the Intensive Care Unit(ICU)

Figure 2b: Specifications of a real time data fusion system in the ICU

Figure 2: Components and specifications of a real time data fusion system in the ICU

1.3 Reference scenario: Sensor fusion for situational assessment

Another example of a complex real time scenario is sensor fusion for situational assessment and command and control [46, 56]. This is the reference scenario that we shall be considering in our evaluations in the rest of the thesis. Sensor fusion is the combining of sensory data or data derived from sensory data from disparate sources such that the resulting information is in some sense better than would be possible when these sources were used individually.

A situational assessment/surveillance scenario comprises a number of entities/objects with complex behaviors moving around in a region. Some platforms (in the case of our reference scenario, these correspond to Unmanned Aerial Vehicles) with sensors are placed across the region. The sensors sense different attributes/features pertaining to the entities/objects in the scenario. The objective of this system is to periodically fuse sensor reports from different platforms into a situational assessment. One connotation of the term “situational assessment” could be the determination of entity types/ identities at different locations in the region. This boils down to a real time tracking and data association problem. Another objective of such a system is to drive command and control platforms/sensors to optimize situational awareness over the region.

Figure 3 illustrates the various components of such a system. The scenario under surveillance comprises a region of interest with a number of moving objects and sensor mounted platforms. One or more sensors are mounted on each platform and each of the sensors detects some feature pertaining to an entity. The platforms periodically deliver a set of sensor reports to a global fusion component, which processes and correlates the data delivered from various sources. Subsequently, based on the information so synthesized by the fusion component, a situational assessment is made. The situational assessment may be in terms of identifying tracks and data associations which are periodically fed back to the human operator or in more complex scenarios may involve updation of plans that drive a command and control algorithm which in turn redirects platforms based on the synthesized information. In either case, the fusion algorithm along with the situational assessment and command

and control components need to keep up with the rate at which sensor reports are periodically delivered to the system in order to ensure a temporally consistent assessment of the scenario. This requires a non-trivial amount of computing power. The system model in the case of this scenario is time triggered because it comprises a set of periodic tasks. The timing constraints are soft because no immediate catastrophe results if a deadline is missed.

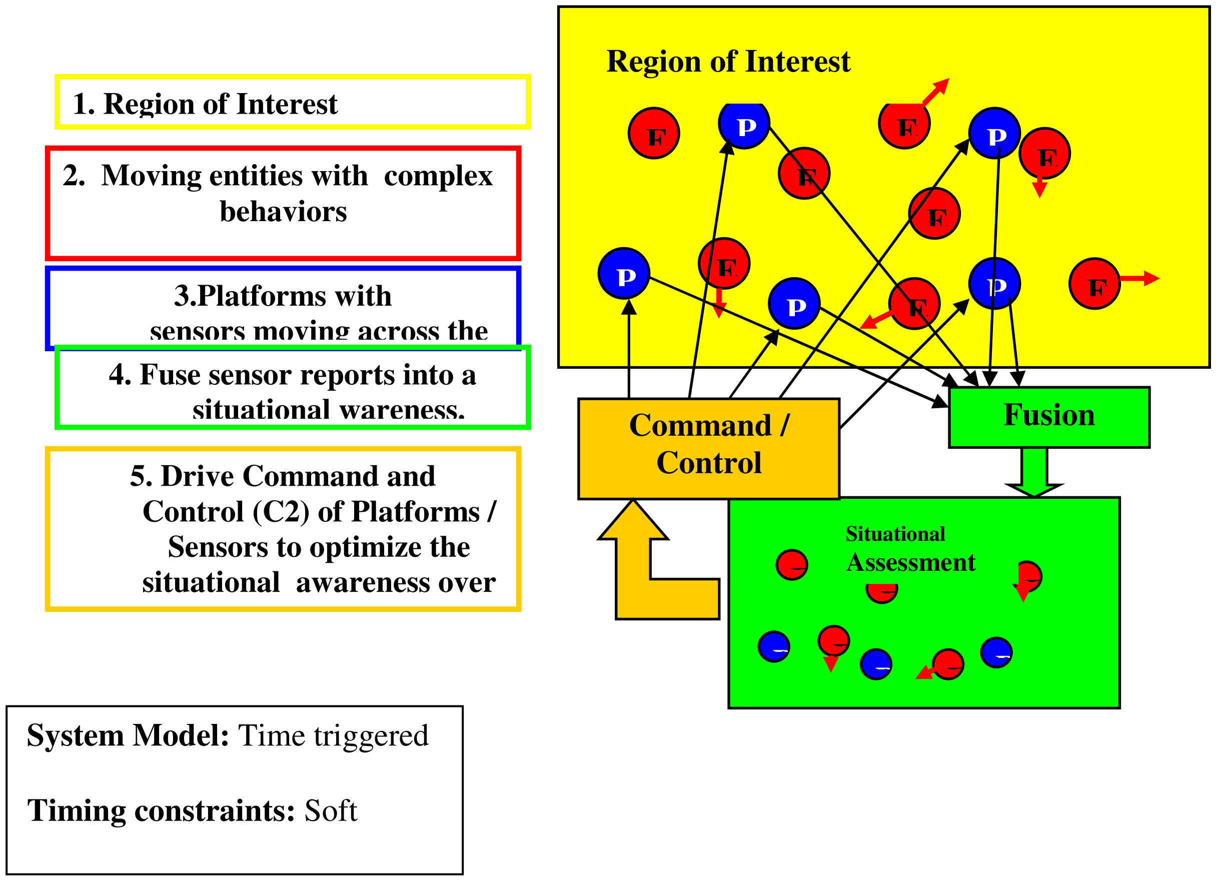

1. Region of Interest
2. Moving entities with complex behaviors
3.Platforms with sensors moving across the
4. Fuse sensor reports into a situational wareness.
5. Drive Command and Control (C2) of Platforms / Sensors to optimize the situational awareness over
System Model: Time triggered
Timing constraints: Soft
Region of Interest
P
F
Command / Control
Fusion
Situational Assessment

Figure 3: Real time situational assessment scenario and system specifications

1.4 Motivation: Soft computations and performance accuracy trade-offs

As discussed in earlier sections, 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. In other words, in the current real time computing paradigm, the objective of the system designer is to ensure that each of the tasks involved in the system always completes execution before a deadline is reached. This, of course, requires a predictable (bounded) model of the environment's impact on the computer system. In the absence of a predictable (bounded) model of the environment, with the use of existing dynamic priority driven scheduling policies, deadlines can be met but not guaranteed and tasks that miss deadlines cannot be predicted.

With the increasing complexity of real time systems, it is becoming infeasible to build systems with sufficient resources to meet the functional and timing requirements of all tasks at all times [39, 40, 41, and 45]. The good news however is that, a growing number of kernels [47, 48, 49], currently offer and several others can be structured to offer [50, 51, 52] a simple means of trading off computation time for the quality of results. A lot of such kernels are particularly prevalent in the domains of artificial intelligence, signal processing and multimedia. We call such kernels "soft kernels" or "soft computations". The chief characteristic of these computations is the fact that they are able to provide cruder results before they complete, or they may execute for a long time refining an already adequate result. In other words, such computations may be able to provide useful/incremental results before fully completing execution.

The idea that such a trade-off can be characterized and used at run time to optimize the performance of complex real time systems is the main motivation of this thesis. Essentially soft computations facilitate meeting real time system deadlines with sufficient system solution quality without having to design the system to support worst case program execution. This is facilitated by the fact that soft computations provide anytime solutions to problems. When designing a system with a number of soft computing kernels, the main issue is ensuring solution quality is sufficient when a deadline arrives rather than supporting the worst-case execution of the program.

The observation that soft computing kernels exist and that their properties can be exploited for real time performance was first made by Liu et. al in [24, 25] . They proposed the imprecise model of computation that uses the strategy of dividing every time-critical task into two logical subtasks: a mandatory subtask and an optional subtask. The mandatory subtask is required for an acceptable result and must be computed to completion before the task deadline. The optional subtask refines the result. It can be left unfinished and terminated at its deadline, if necessary, lessening the quality of the task result. We shall discuss the major differences between our work and the work done by Liu et.al [24, 25] in Chapter 3.

In order to understand the opportunities provided by soft computing kernels to improve real time performance, let us look at Figures 4 and 5 more carefully. Figure 4 shows typical computation time Vs quality (time-value) functions for different types of computations. Figure 4a shows the time-value function for a traditional algorithm.

What this curve essentially illustrates is the fact that a traditional algorithm has just one output /solution which can be obtained provided a sufficient amount of time is provided to the task for execution. If sufficient amount of time is not allocated to the algorithm, no output/solution is obtained. Figures 4b, 4c and 4d are examples of time-value functions for soft computations. What these figures illustrate is that when using soft computing kernels, output/solutions of lesser quality/value are obtainable even if execution of the kernels is restricted to a fraction of the time required for the computation to deliver the best possible value/solution to the problem.

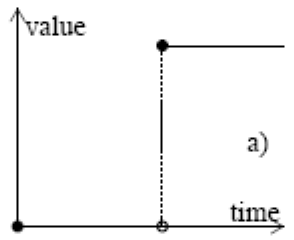


Figure 4a Traditional algorithm

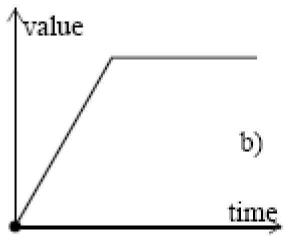


Figure 4b Soft computation: Type 1

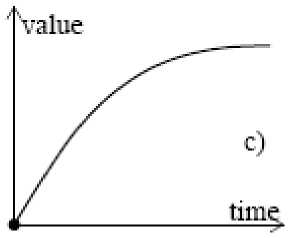


Figure 4c Soft computation: Type 3

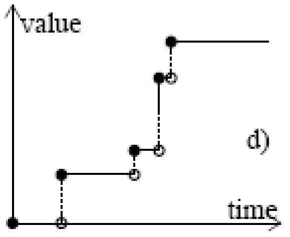


Figure 4d Soft computation: Type 4

Figure 4: Functions representing the variation of solution quality with computation time for traditional algorithms and those having soft computing properties

Figure 5 illustrates how soft computing kernels can facilitate efficient design of real time systems by eliminating the need to design the system to support worst case program execution. Figure 5a shows a traditional algorithm that is unable to produce a solution at the deadline (represented by the dotted line) in the absence of sufficient resources to meet its processing needs. It produces a solution beyond the deadline at which point of time the solution has lesser utility for the system(if the deadline is soft) or causes catastrophic results (if the deadline is hard). Figure 5b illustrates how a soft computing kernel would behave in a similar scenario. At the deadline, the soft computing kernel provides a solution/output with lesser quality than the maximum achievable by it. But the quality of the solution obtained at the deadline is sufficient to meet the system specifications. Thus by meeting the system deadline with a solution of sufficient quality, under resource constraints, the soft computation can avoid the catastrophic consequences/decreased system utility resulting from a traditional computation completing and providing an output beyond the deadline.

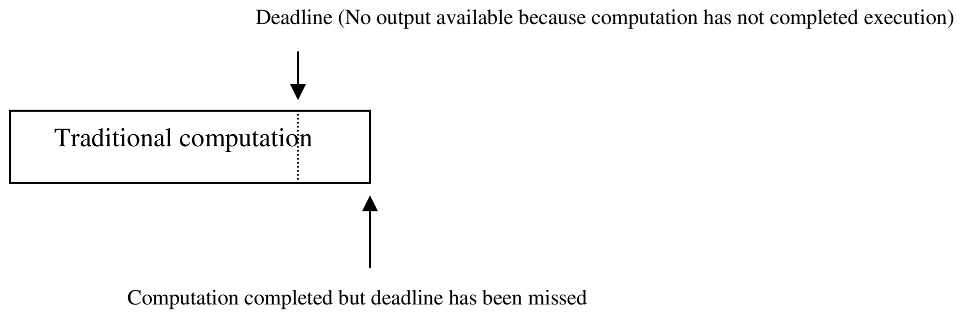

Deadline (No output available because computation has not completed execution)
Traditional computation
Computation completed but deadline has been missed

Figure 5.a. A traditional algorithm in a resource constrained scenario does not provide a solution at the deadline .

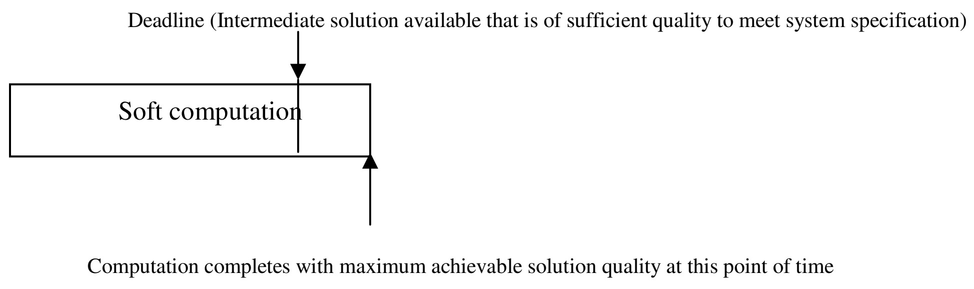

Deadline (Intermediate solution available that is of sufficient quality to meet system specification)
Soft computation
Computation completes with maximum achievable solution quality at this point of time

Figure 5.b A "soft computation" under resource constraints provides solutions with sufficient quality at the deadline

In addition to the fact that it is becoming infeasible to design complex real time systems with sufficient resources to meet all system deadlines, the motivation for incorporating computation utility/value based run-time decisions is further accentuated by the following two observations about the behavior of conventional static and dynamic real time scheduling schemes:

- Static schemes 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 work), the average resource utilization is low. Hence, there is considerable scope for value-added computation.
- Though current dynamic schemes provide better resource utilizations, they react unpredictably to failures and overloads .Static schemes may be able to cope with certain failures (as defined in its failure model), but once the system moves outside its failure model , no level of service can be depended upon. Hence, it is desirable to allow graceful degradation (of service) when resources are scarce.

1.5 Thesis contributions and organization

In the presence of a conglomeration of traditional “non-soft” and “soft computing” kernels (as defined in section 1.4) in a real time system, one can envision adopting a completely different/novel approach to the design of real time systems. Instead of trying to design a system that would meet a specific set of timing constraints, the design problem can be divided into three orthogonal issues:

- decomposition of the total system into performance components
- implementation of as many of these basic components as “soft kernels “

- formulation of a model of the quality of functional correctness of the system as a function of various compositions of the software components. Such a model can subsequently be used to dynamically schedule tasks and decide the optimal composition of basic tasks/components at run time.

In this thesis, we try to answer a few questions pertaining to this design philosophy in the context of a specific complex real time scenario [56]. More specifically, we attempt to increase the flexibility of real-time systems by allowing certain decisions about the system's behavior like the composition of "soft" tasks/kernels to be made at run-time. This requires some form of dynamic scheduling. The need to support value-added computation and graceful degradation is necessarily complicated by the reality that not all services have equal utility at all times. We use models that characterize the performance accuracy trade-offs of kernels to control the run-time decision process and determine the optimal composition of soft tasks given a time constraint /deadline.

The work done as a part of this thesis uses the real time system/scenario described in section 1.3 as the reference scenario. The system prototype of this scenario comprises the following:

1. A parameterized simulation framework/ test bed [55] provided by Northrop Grumman Corporation that simulates a defense surveillance scenario with moving entities having complex behaviors and a number of platform mounted sensors monitoring them.
2. The Information Matrix Data Association algorithm developed by Schumitsch et.al [58] to solve the complex tracking and data association problem in the

context of sensor fusion. The MATLAB version of the code provided by the authors was converted to C so that it could be used for our study.

In this thesis, we try to demonstrate the usefulness of the design philosophy for real time systems described above through the following contributions:

1. We build a system prototype of the real time scenario and application (real time sensor fusion and situational assessment) mentioned in section 1.3 and discussed in detail in Chapter 4. The development of this system prototype involved:

a. Porting the MATLAB code of the Information Data Association algorithm provided by the authors of [58] to C.

b. Integrating and applying the sensor fusion algorithm to a cognitive test bed, provided by Northrop Grumman that simulates a situational assessment scenario.

c. Building a preliminary framework for investigating different real time scheduling policies in the context of the system prototype and the reference scenario.

2. We thereafter identify "soft computations" in the system prototype of the real time scenario under consideration. We subsequently characterize the performance-accuracy tradeoffs of individual "soft kernels"/"soft computations" that form a part of the system using performance profiles.

3. We propose and demonstrate the benefits of a methodology of using performance profiles to build models that can be used at runtime by a dynamic scheduler for determining the optimal composition of tasks so as to maximize the utility of the system given a specific limited amount of time to meet the system objective. This answers the question about how much time should be allocated to each of the components of the real time system to maximize its overall utility.

4. We quantify the benefits of using a performance profile based model for scheduling a mixture of traditional “non-soft” and “soft computing” tasks as opposed to using conventional static scheduling and dynamic Earliest Deadline First (EDF) schemes. We quantify the gains both in terms of the functional correctness of the system, the CPU utilization and the number of missed deadlines.

5. We quantify how the variance in the dynamics/performance of the “non-soft” kernels of the system justifies the use of run-time monitoring rather than determining a fixed running time/composition of “soft” tasks when the system is activated. We quantify the gains both in terms of the functional correctness of the system and the CPU utilization.

6. We also perform a sensitivity study to determine how the models derived from performance profiles change with changes in various input parameters of the datasets and identify the potential benefits of learning performance profiles online.

The rest of this thesis is organized as follows. Chapter 2 describes background on real time systems. Chapter 3 discusses related work in the area of real time scheduling. Chapter 4 describes the system prototype that has been built. This includes description of the reference real time problem scenario, the sensor fusion algorithm and the test bed /datasets used in the experiments. Chapter 5 explains the methodology used to characterize soft computations and performance-accuracy trade-offs. Chapter 6 discusses the methodology for run-time scheduling of soft computations using models derived from performance profiles. It also describes the policies with which our scheme has been compared. Chapter 7 discusses our experimental results. Chapter 8 summarizes and concludes our work.

Chapter 2

Background and Terminology: Real time Systems

In this chapter, we introduce and discuss the basic concepts and terminology in the real time systems domain.

2.1 Real time computing

Real time computing systems are defined as those systems in which the correctness of the system depends not only on the logical results of the computation, but also on the time at which they are produced [53]. The objective of real time computing is to meet the timing and functional requirements of individual tasks. Additionally it is also desirable that real-time systems achieve their functional correctness and timeliness while being highly utilized.

As an example of a conventional real-time system, consider a computer-controlled machine on the production line at a bottling plant. The machine's function is simply to cap each bottle as it passes within the machine's field of motion on a continuously moving conveyor belt. If the machine operates too quickly, the bottle won't be there yet. If the machine operates too slowly, the bottle will be too far along for the machine to reach it. Stopping the conveyor belt is a costly operation, because the entire production line must correspondingly be stopped. Therefore, the range of motion of the machine coupled with the speed of the conveyor belt establishes a window of opportunity for the machine to put the cap on the bottle. This window of opportunity imposes timing constraints on the operation of the machine. Applications with these kinds of timing constraints are considered real time. In this case, the timing constraints are in the form of a period and deadline.

2.2 Deadlines and Periods

The period is the amount of time between instances of a regularly repeated task. Such repeated tasks are called periodic tasks. For instance in the context of the example mentioned earlier, suppose bottles pass under the machine at a rate of five per second. This means a new bottle shows up every 200ms. Thus, the period of the task is 200ms. Note that whether bottles pass once per second or 100 times per second, it doesn't change the fact that this is a real-time system. Real time does not mean fast; it means that a system has timing constraints that must be met to avoid failure.

The deadline is a constraint on the latest time at which the operation must complete. Suppose the window of opportunity is 150ms. The deadline is then 150ms after the start time of the operation. In our example, the start time is defined as the moment the bottle enters the range of the machine. This bottle example has physical constraints, namely the speed of the conveyor belt and the machine's range of motion, that dictate the period and deadline of the task.

In many real-time systems, the period is a design parameter. Consider a cruise control mechanism on an automobile. The basic operation of cruise control is to keep the speed of the vehicle constant. Suppose the driver selects 60mph as the desired speed. If the vehicle is going slower than 60mph, then the embedded computer sends a signal to the engine controller to accelerate. If the vehicle is going faster than 60mph, it sends a signal to decelerate. A question to ask is: how often does the computer check if the current speed is too slow or too fast? The answer is called the control rate (or frequency). It is defined by the control system designer, who will try to find a rate that is fast enough to meet specifications, but not so fast that it adds unnecessary cost to the system. The period is then the reciprocal of the rate (that is, period = 1/rate). The deadline is typically the beginning of the next cycle of a periodic task, because, to start the new cycle, it needs to be finished with the old one.

Communication systems also have real-time constraints. Suppose a multimedia application needs to compress video data at a rate of 30 frames per second. Before it processes a new frame, it needs to finish processing the old frame, otherwise data might get lost in the form of dropped frames. The period of such a task is the frame rate. Processing the old frame must complete before processing on the new frame can begin. Therefore, the deadline is the beginning of the next frame.

2.3 Handling Aperiodic tasks

Not all real-time tasks are periodic. Aperiodic tasks, also called aperiodic servers, respond to randomly arriving events. Consider anti-lock braking. If the driver presses the brake pedal, the car must respond very quickly. The response time is the time between the moment the brake pedal is pressed, and the moment the anti-lock braking software actuates the brakes. If the response time was one second, an accident might occur. So the fastest possible response is desired. But, like the cruise control algorithm, fastest is not necessarily best, because it is also desirable to keep the cost of parts down by using small microcontrollers. What is important is for the application requirements to specify a worst-case response time. The hardware and software is then designed to meet those specifications.

Note that many aperiodic tasks can be converted to periodic tasks. This is basically the same transformation as converting an interrupt handler to a polling task. Instead of reacting to an external event the moment it occurs, the software polls the external input regularly, perhaps tens or hundreds of times per second. If the awaited event is detected, the appropriate computation is enacted.

2.4 System Models

There are two general paradigms for the design of real-time systems known as Time-Triggered (TT) and Event-Triggered (ET) architectures [53], both of which are explained next.

- System activities in TT are initiated at predefined instants, and therefore TT architectures require the assessment of resource requirements and resource availability prior to the execution of each application task. Each task's needed resources and the length of time over which these resources will be used can be computed off-line in a resource requirement matrix. If these requirements cannot be anticipated, then worst case resource and execution time estimates are used. Thus, TT is prone to wasted resources and lowered system utilization since resource requirement estimates are pessimistic. However, TT architecture can provide predictable behavior due to its pre-planned execution pattern.
- System activities in ET are initiated in response to the occurrence of particular

events that are possibly caused by the environment. In ET architectures, an excessive number of possible behaviors must be carefully analyzed in order to establish their predictability, because resource needs and availability may vary at run-time. Thus, the resource-need assessment in ET architecture is usually probabilistic. Although, ET is not as reliable as TT architecture, it provides more flexibility and is ideal for more classes of applications, which do not lend themselves to predetermination of resource requirements.

2.5 Hard and soft real time systems

Real-time applications can be modeled as a set of tasks, where each task can be classified according to its timing requirements as hard or soft. A hard real-time task is the one whose timely and logically correct execution is considered to be critical for the operation of the entire system. In a hard real-time system, if one or more activities miss a deadline or timing constraint, the system fails. Failure includes damage to the equipment, major loss in revenues, or even injury or death to users of the system. One example of a hard real-time system is a flight controller. If action in response to new events is not taken within the allotted time, it could lead to an unstable aircraft, which could, in turn, lead to a crash.

The deadline associated with a hard real-time task is conventionally termed a hard-deadline. Since missing a hard-deadline can result in catastrophic consequences; such systems are known as safety-critical. Thus, the design of a hard real-time system requires that a number of performance and reliability trade-off issues to be carefully evaluated.

In contrast, a soft real-time system is one that has timing requirements, but occasionally missing them has negligible effects, as application requirements as a whole continue to be met. Consider again the cruise control application. Suppose the software fails to measure current velocity in time for the control algorithm to use it. The control algorithm can still use the old value, because the amount that the velocity would have changed between the last sample and this sample is so small that it can still operate correctly. Missing several consecutive samples, on the other hand, could be a problem, as the cruise control would likely stop meeting application requirements because it is not able to maintain the desired speed within a proper error tolerance. Thus soft real-time application is characterized by a soft-deadline whose adherence is desirable, although not critical, for the functioning of the system. That is, missing a soft-deadline does not cause a system failure or compromise the system's integrity. There may still be some (diminishing) value for completing an application after its deadline, without any catastrophic consequences resulting from missing such a deadline.

The distinction between a soft and a hard real time system is however somewhat fuzzy [53]. As illustrated in Figure 6, the meaning of real-time spans a spectrum. At one end of the spectrum is non-real-time, where there are no important deadlines (meaning all deadlines can be missed). The other end is hard real-time, where no deadlines can be missed. Every application falls somewhere between the two endpoints.

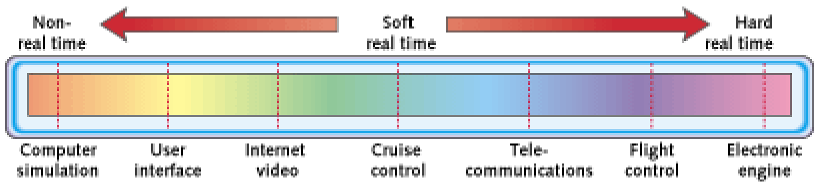


Figure 6: The real time spectrum [53].

2.6 Types of real time tasks

As a direct consequence of these timing requirements and the system models we mentioned earlier, real time application tasks can be classified as periodic, aperiodic, or sporadic tasks.

1. Periodic tasks are those tasks that execute at regular intervals of time i.e. every ‘T’ time units – corresponding to TT architectures. These tasks typically tend to have hard deadlines, characterized by their period(s) and their required execution time per period, which is usually given, by a worst-case execution time.

2. Aperiodic tasks are those tasks whose execution time cannot be anticipated apriori. This means the activation of aperiodic tasks is essentially a random event caused by a trigger – corresponding to ET architectures. Such a behavior does not allow for worst-case analysis, and therefore aperiodic tasks tend to have soft deadlines.

3. Sporadic tasks are those tasks that are aperiodic in nature, but they have hard deadlines. Such tasks can be used to handle emergency conditions and/or exceptional situations. Due to the nature of hard deadlines, worst-case calculations may be facilitated by a schedulability-constraint [54, 57], which defines a minimum period between any two sporadic events from the same source. Such tasks are converted to

periodic tasks by basically using the same transformation as converting an interrupt handler to a polling task.

2.7 Predictability and Determinism

Two more terms often used to describe real-time systems are predictable and deterministic. These terms are related, but because they are often interchanged, they are often a source of confusion.

Real-time systems researchers generally use the term predictable to refer to a system whose timing behavior is always within an acceptable range. The behavior is specified on a system-wide basis, such as "all tasks will meet all deadlines." Generally, the period, deadline, and worst-case execution time of each task need to be known to create a predictable system.

A deterministic system is a special case of a predictable system. Not only is the timing behavior within a certain range, but that timing behavior can be pre-determined. For example, a system can be designed with pre-allocated time slots for each task. Execution for each task occurs only during those time slots. Such a system must have execution time for every task known, as well as no anomalies that might cause deviation from the pre-determined behavior. That is, of course, difficult to achieve. Fortunately, determinism is not essential to build predictable real-time systems.

2.8 Real time scheduling

When activities have timing constraints, as is typical of real time computing systems, scheduling these activities to meet their timing constraints is one major problem that comes to mind. Traditional real time scheduling algorithms fall into two categories: static and dynamic.

- A static/pre-runtime approach calculates (or predetermines) schedules for the system off-line. It requires prior knowledge of all task characteristics (arrival times, deadlines, release times and worst case execution times).
- In a dynamic or runtime approach, the order of execution of tasks is decided at runtime based on priorities attached to tasks. These priorities in turn may be determined off-line (like in the case of the rate monotonic algorithm [1] where priorities are based on the frequency/periods of tasks) or online (like in the case of the Earliest Deadline First algorithm [1], where a task with the earliest deadline is given highest priority).

Certainly in safety critical systems it is reasonable to argue that no event should be unpredicted and that schedulability should be guaranteed before execution [5]. This implies the use of a static scheduling algorithm or at least a static priority driven algorithm. Dynamic priority driven approaches do, nevertheless, have an important role particularly in soft real time systems or in applications where guaranteed worst case execution time /scenario is not possible.

Chapter 3

3. Related Work

In this chapter, we discuss prior research related to our work. Our work is primarily related to research in the area of real time scheduling. Within the context of real time scheduling, we discuss work in conventional real time scheduling techniques [1-23] as well as real time scheduling techniques based on imprecise computations [24, 25, 34, 36] .

The landmark paper in the arena of real time scheduling has been from Liu et.al [1], where the authors theoretically prove the optimality of two priority driven scheduling algorithms for real time systems scheduling. They propose the Rate Monotonic (RM) algorithm, which is a static priority driven algorithm that assigns priorities to tasks based on the size of their periods. The tasks with smaller periods are assigned higher priorities. These task priorities are then used at run time to make a decision about which task is executed at a particular point of time. The other algorithm the authors propose in [1] is the Earliest Deadline First (EDF) algorithm, which is a dynamic priority driven algorithm where the task with the earliest deadline is assigned the highest priority at any point of time and is chosen for execution. Both these proposed algorithms have been mathematically proven to be optimal under the following constraints of the task model:

- All tasks under consideration are periodic
- All tasks are preemptive

- Deadlines consist of run-ability constraints only-i.e. each task must be complete before the next request for it occurs.
- The tasks are independent in that requests for a certain task do not depend on the initiation or completion of other tasks.
- Run time for each task is constant work.

This classical paper on real time scheduling was followed by various others which proposed algorithms /heuristics based on these two algorithms but differed in one of the following aspects:

1. **Priority assignment schemes:** Liu et.al.[1] proposed an algorithm called

Minimum Laxity First(MLF) that assigns a *laxity* to each task at run time. At any point of time, the scheduler selects the task with the minimum laxity to execute. Laxity is defined as: laxity = (deadline time - current time - CPU time needed). A laxity of t_i means that even if the task is delayed for t_i time units, it will still meet its deadline. A laxity of '0' means that the task must be executed *now* or will fail to meet its deadline. Main difference between MLF and EDF is that MLF takes into consideration the *execution time* of a task. Like EDF, MLF has a schedule bound of 100% and there is no way to guarantee *which* task(s) will fail in transient overload. The analysis of MLF is also based on the same task model as EDF and RM algorithms. In [22], Stewart et.al propose the Maximum Urgency First(MUF) heuristic which is a combination of fixed and dynamic priority scheduling (a.k.a. *mixed priority*). Each task is assigned an *urgency* which is defined as a combination of two fixed priorities and one dynamic priority. One of the fixed priorities, *the criticality*, has precedence over the dynamic priority. Meanwhile, the dynamic priority has precedence over the other fixed priority, called the *user priority*. The dynamic priority is inversely proportional to the laxity of a task. MUF consists of two parts. The assignment of the criticality and user priority (done *apriori*), and the actions of the MUF scheduler done at run-time. Note that static priorities are assigned once and do not change during execution. The dynamic priority is assigned at run-time, inversely proportional to the laxity. The task model for the analysis of the MUF algorithm is the same as the one used by Liu et. al.[1]. In [21], Salmani et.al proposed the Modified Maximum Urgency First (MMUF) heuristic. In this scheme, they use earliest deadline first and modified least laxity first algorithms for calculating the dynamic priorities of the MUF algorithm.

2. **Assumed task model:** In [16] , Li et.al propose *group-EDF* (gEDF), a heuristic for scheduling soft non-preemptive tasks and is based on dynamic grouping of tasks with deadlines that are very close to each other. They use *Shortest Job First* (SJF) technique to schedule tasks within the group. They were motivated by the belief that grouping tasks dynamically with similar deadlines and utilizing a secondary criteria, such as minimizing the total execution time (or other metrics such as power or resource availability) for scheduling tasks within a group, can lead to new and more efficient real-time scheduling algorithms. In [20], Buttazzo et.al propose a set of heuristics based on Earliest Deadline First for scheduling soft aperiodic tasks. In [54], Rajkumar et.al extend rate-monotonic scheduling theory to periodic tasks that are not independent, but must contend for exclusive access to shared resources.

3. **Handling of overload conditions:** In [8, 9, 12, 13, 18], variations of the rate

monotonic and earliest deadline first algorithms have been proposed for handling overload conditions in real time systems

There have been a lot of other heuristics/algorithms [10, 14, 15, 16, 17, 19, 20] proposed in the conventional real time scheduling paradigm. Unlike conventional real time scheduling techniques like Rate Monotonic and Earliest Deadline First, our dynamic scheduling scheme does not take scheduling decisions based on task priorities. Instead our scheduling scheme uses performance profile based models to determine the optimal composition of tasks at any point of time to maximize system utility at a deadline.

Our work is closest to the research on scheduling schemes based on the imprecise computation model. The imprecise computation model was proposed by Liu et.al in [24, 25] .The imprecise computation technique uses the strategy of dividing every time-critical task into two logical subtasks: a mandatory subtask and an optional subtask. The mandatory subtask is required for an acceptable result and must be computed to completion before the task deadline. The optional subtask refines the result. It can be left unfinished and terminated at its deadline, if necessary, lessening the quality of the task result. The result produced by a task when it completes is the desired precise result, which has an error of zero. If the task is terminated before completion, the intermediate result produced at that point is usable as long as the mandatory subtask is complete. Such a result is said to be imprecise. The system schedules and executes to completion all mandatory tasks before their deadline but may leave less important optional tasks unfinished if necessary.

In [36], Stankovic et.al have examined the performance of various scheduling policies for managing transient overload in an imprecise computation system. If the load on the computation system is low, the scheduler is designed to provide some prescribed balance of accuracy and response time. If the load is high, the scheduler is designed to keep response time bounded by sacrificing accuracy.

The main differences between the scheduling approach proposed by us and researchers working on the imprecise computation model are:

1. **Objective characterization of the quality of soft computations:**

We objectively quantity the performance-accuracy trade-offs of "soft computations" in the form of performance profiles by measuring a concrete, well-defined aspect of the quality of the results of soft computations as a function of execution time. The performance profiles are obtained through calculations of concrete metrics rather than human intuition. To the best of our knowledge, the proposed "precision value" functions of the imprecise computation model are subjective [24, 25, 27, 38].

2. Composition of tasks

The imprecise computation model does not address the issue of optimal composition of tasks given a specific time constraint. Instead, it deals with individual, independent tasks. Since the problem definition imposes timing constraints on the tasks, it allows the tasks to be *temporally dependent*, but it assumes that the results of each task and their qualities are independent. This is a major simplification that cannot be made when dealing with a system where the goal is to achieve a system objective (as in the case of complex real time systems) with a number of interdependent tasks.

Our task model assumes that a real time application is composed of individual software modules/tasks, many of which may be interdependent. Tasks are a part of a single application with a system wide objective. Our objective is to maximize the value of a particular composition of tasks at a deadline so as to meet the system objective in the best possible way given time and resource constraints. In simple terms, our scheduling scheme addresses a decision problem involving the choice of a collection of services to execute so the `best possible' outcome ensues for the system. At various decision points (at run-time) there are a set of tasks/services that are available for execution. Unfortunately there may not be enough resources to execute all services to completion. 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 and to what extent when resources are scarce. The issue of determining the optimal composition of tasks so as to maximize the value of the system objective is addressed in our work.

3. To the best of our knowledge, the benefits of conventional scheduling techniques as well as those based on the imprecise computational model have been demonstrated on randomly generated, periodic, independent task sets. We demonstrate our methodology on a real application and a real scenario. The downside of this is the fact that some of our assumptions become application specific. However, we believe that this is a reasonable first step for realistically demonstrating the benefits we claim from our work.

Another important area related to the work we've done is the arena of anytime algorithms. The derivation of performance profiles for individual "soft kernels" / "soft computations" used in our evaluations is based on similar techniques suggested by Zilberstein et.al in [50, 51, 52]. In these papers, Zilberstein et. al talk about various issues in the context of using anytime algorithms in intelligent systems.

Chapter 4

4. Detailed Problem Formulation

As mentioned in section 1.3, in this thesis we consider the real time sensor fusion and situational assessment challenge problem posed by Northrop Grumman Corporation as a part of an ongoing DARPA project. Here we provide a more detailed description of the problem scenario and the components of the system prototype we have built based on this scenario. First we describe the UAV sensor fusion challenge problem posed by Northrop Grumman. Thereafter, we describe the system prototype used for evaluating solutions to the challenge problem. In this context, we describe Northrop Grumman's cognitive sensor fusion simulation test bed, the Information Data Association Algorithm and the application of this algorithm to the test bed data. Finally we discuss the real time problem formulation pertaining to this challenge problem.

4.1 Reference real time scenario: Northrop Grumman's challenge problem for sensor fusion and situational assessment

The real time sensor fusion problem posed by Northrop Grumman (introduced in section 1.3) [56] captures the general difficulty of processing the large number of reports from a distributed dynamic system and processing those reports to determine a unified situational assessment in real time. This class of problems stresses processing techniques with large numbers of reports to be processed and a strong variability in processing from one instance to the next. The difficulty in this process is to accurately fuse reports and satisfy real-time information needs. In most scenarios, critical decisions must be made within seconds, not minutes or hours. The problem is further complicated by the high-speed feedback loop required to drive optimal sensing, based on the quality of the overall fused situational awareness.

The challenge problem posed has two parts to it. The sub problem we deal with comprises a system with a large number of entities (objects) with complex behaviors moving in a region. Some sensor mounted Unmanned Aerial Vehicle (UAV) platforms also move around in the region attempting to gather data about the entities. The problem is to fuse the large number of sensor reports into a situational assessment of the region in terms of tracking entity types and identities by solving the data association problem. Here the real time constraint as shown in Figure 7 is to periodically provide a situational assessment to a human operator in terms of associations of tracks with entity identities. This is the scenario we are dealing with in this thesis. In order to meet the user defined real time constraint and provide a temporally consistent situational assessment, the global fusion algorithm has to keep up with the rate at which sensor reports are delivered so as to maximize the utility of the system objective at each user defined deadline.

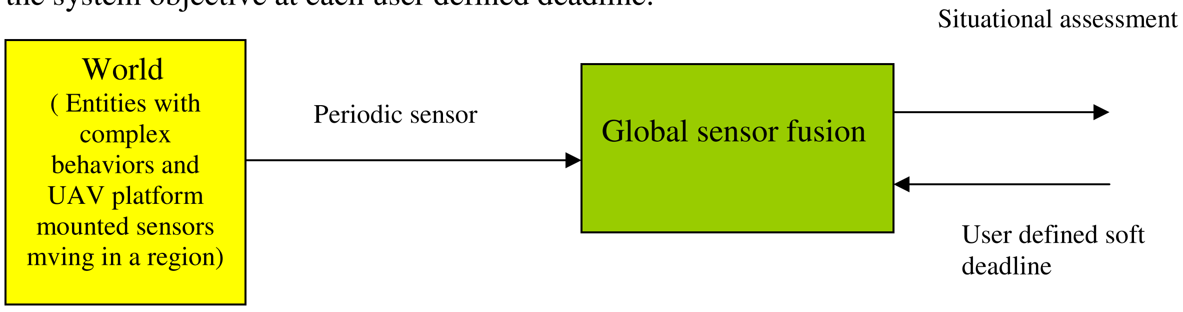

World
(Entities with complex behaviors and UAV platform mounted sensors mving in a region)
Periodic sensor
Global sensor fusion
Situational assessment
User defined soft deadline

Figure 7: Real time reference problem scenario: UAV sensor fusion for situational assessment

The system prototype of the scenario shown in Figure 7 comprises the following components:

- The world is modeled by the Northrop Grumman's cognitive test bed (described in detail in section 4.2.2)
- The global fusion component is implemented using the C version of the Information Form Data Association algorithm [58] (described more in detail in section 4.2.1)

4.2 System prototype

In this section, we discuss the main components of the system prototype of the real time scenario described in section 4.1

4.1.1 Information Form Data Association Algorithm (IDA)

The Information Form Data Association algorithm [58], addresses the problem of data association in online object tracking. The data association problem arises in a large number of application domains, including computer vision, robotics, and sensor networks. This algorithm forms the core of the sensor fusion component of the real time scenario discussed in this thesis.

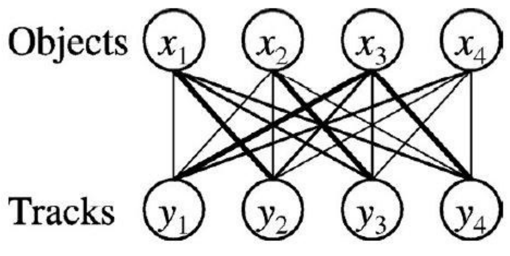


Figure 8 a. Tracking N objects poses N! possible Associations

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Figure 8.b A permutation matrix representing one of 4! possible associations when N=4

Figure 8: Illustration of the data association problem for 4 objects and 4 tracks

Given the problem of associating N objects with N tracks as shown in Figure 8a, the standard probabilistic solution requires exponential update time and exponential memory. This is because each data association hypothesis is expressed by a permutation matrix as shown in Figure 8b that assigns computer-internal tracks to objects in the physical world. An optimal filter would therefore need to maintain a probability distribution over the space of all permutation matrices, which grows exponentially with N, the number of objects in the world. This essentially means that to do a full Bayesian solution, we must maintain the posterior probabilities p(A) of N! permutation matrices represented by *A*.

The common remedy involves the selection of a small number K of likely hypotheses. This is the core of numerous widely used multi-hypothesis tracking algorithms [59]. More recent solutions involve particle filters [60], which maintain stochastic samples of hypotheses. Both of these techniques are very effective for small N, but the number of hypothesis they require grows exponentially with N.

The Information Form Data Association algorithm (IDA) [58] is a filter algorithm that scales to much larger problems. This filter maintains an information matrix (henceforth represented by Ω) of size $N \times N$ (where N is the number of objects in the world), which relates tracks to physical objects in the world. The rows of Ω correspond to object identities, the columns to the tracks of the tracker. Ω is a matrix in information form, that is, it can be thought of as a normalized log-probability.

The key innovation is a representation of the data association posterior in information form, in which the "proximity" of objects and tracks are expressed by numerical links. Updating these links requires linear time, compared to exponential time required for computing the exact posterior probabilities.

This algorithm assumes an online tracking system that receives sensor data, conveying information about the identity or type of objects that are being tracked. The algorithm also models the uncertainty introduced through the tracker's inability to reliably track individual objects over time due to mixing of tracks.

As mentioned earlier, the central data structure in the Information Form Data Association algorithm is the information matrix (Ω). This matrix encodes the probability distribution across the different possible permutations/associations of objects with tracks.

The three main subparts of the algorithm are:

1. Mixing Updates: This part of the algorithm models the uncertainty in the system due to tracks being very close to each other

$$\begin{pmatrix} 2 & \boxed{12} & \boxed{4} & 4 \\ 1 & \boxed{2} & \boxed{11} & 0 \\ 10 & \boxed{4} & \boxed{4} & 15 \\ 5 & \boxed{3} & \boxed{1} & 2 \end{pmatrix} \longrightarrow \begin{pmatrix} 2 & \boxed{11.31} & \boxed{11.31} & 4 \\ 1 & \boxed{10.31} & \boxed{10.31} & 0 \\ 10 & \boxed{4} & \boxed{4} & 15 \\ 5 & \boxed{2.43} & \boxed{2.43} & 2 \end{pmatrix}$$

Figure 9: Representative mixing update of the Information Matrix (Ω)

Closeness/mixing of tracks causes a loss of information with respect to the data association. The tracker confusing two objects amounts to a random flip of two columns in the data association matrix/permutation matrix represented by A. Let {B1, B2,. . . ,Bm} be a set of permutation matrices, and {β1,β2,β3,β4,...βm} be a set of associated probabilities. The "true" permutation matrix undergoes a random transition from A to ABm with probability βm. This corresponds to the following update rule in the information matrix corresponding to the mixing of two tracks:

$$\Omega \quad \longleftarrow \quad \log \sum_m \beta_m \, B_m^T \, \exp \Omega$$

Here the expression “exp” denotes a component-wise exponentiation of the matrix; the result is also a matrix. This update implements a “dual” of a geometric mean; here the exponentiation is applied to the individual elements of this mean, and the logarithm is applied to the result. It is important to notice that this update only affects elements in Ω that might be affected by a permutation Bm; all others remain the same. In other words, if affects only those columns corresponding to tracks that mix.

2. Observation Update: This part of the algorithm updates the information matrix based on local observation data about objects.

$$\begin{pmatrix} 2 & 12 & 4 & 4 \\ 1 & 2 & 11 & 0 \\ 10 & 4 & 4 & 15 \\ 5 & \boxed{2} & 1 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 2 & 12 & 4 & 4 \\ 1 & 2 & 11 & 0 \\ 10 & 4 & 4 & 15 \\ 5 & \boxed{3} & 1 & 2 \end{pmatrix}$$

Figure 10: Representative update operation for an observation that links track 2 to object 4

The update rule employed by the algorithm for performing observation updates on the information matrix is as follows:

If probability of object ‘i’ being track ‘j’ given sensor reading zj = zij i.e.

$$p(x_i = y_j \mid z_j) \quad = \quad z_{ij} \qquad \text{with} \quad \sum_i z_{ij} = 1$$

then

$$\Omega \longleftarrow \Omega + \begin{pmatrix} 0\cdots 0 & \log z_{1j} & 0\cdots 0 \\ \vdots\ddots\vdots & \vdots & \vdots\ddots\vdots \\ 0\cdots 0 & \log z_{1N} & 0\cdots 0 \end{pmatrix}$$

After each update, a normalization operation is performed to make sure that all elements in Ω are positive.

3. Inference: This part of the algorithm infers the data associations of objects with tracks

$$\begin{pmatrix} \mathbf{2} & 12 & 4 & 4 \\ \mathbf{1} & 2 & 11 & 0 \\ \mathbf{10} & 4 & 4 & 15 \\ \mathbf{5} & 2 & 1 & 2 \end{pmatrix}$$

Figure 11a: Example Information Matrix (Columns= tracks; Rows=objects)

$$\begin{pmatrix} 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} \\ \mathbf{1} & 0 & 0 & 0 \end{pmatrix}$$

Figure 11b: Permutation matrix representing optimal data association

Using the information matrix shown in Figure 11a, if we were to solve the data association problem using a maximum likelihood formulation, we would look at each column individually, pick the maximum value corresponding to that column and then consider the row index corresponding to the maximum value to be the correct data association to the track represented by the column under consideration. For instance, by looking at each of the four columns corresponding to the information matrix represented in Figure 11a in isolation, with a maximum likelihood formulation for inferring track-object associations, it would appear that track 1 should be associated with object 3, track 2 to object 1, track3 to object 2 and track 4 to object 3 again. However this solution associates both tracks 1 and 4 to object 3, which is incorrect. Figure 11b represents the permutation matrix representing the optimal data association, where track 1 is associated with object 4, track 2 with object 1, track 3 with object 2 and track 4 with object 3. The objective of the inference part of the algorithm is to obtain the optimal data association from the information matrix. This optimal data association can be performed using a simulated annealing formulation which is done in our system prototype..

4.2.2 Northrop Grumman's cognitive test bed

Northrop Grumman's UAV sensing cognitive test bed environment is a non-real time simulation environment designed to allow the opportunity to flexibly test and characterize sensor fusion and command and control algorithms within the full context of the dynamic UAV sensing problem. The test bed contains functionality to wrap an algorithm with interfaces to sensors and platforms and to allow the algorithm to track its own situational awareness. Behind the scenes, a behavioral model of the physical entities, platforms, and sensors drive the simulation. The functional correctness of the various algorithms applied to this test bed is determined through comparison to the simulation truth over time. Logging capabilities allow the test bed to generate data sets for external testing and analysis. Figure 12 shows the main components of the test bed.

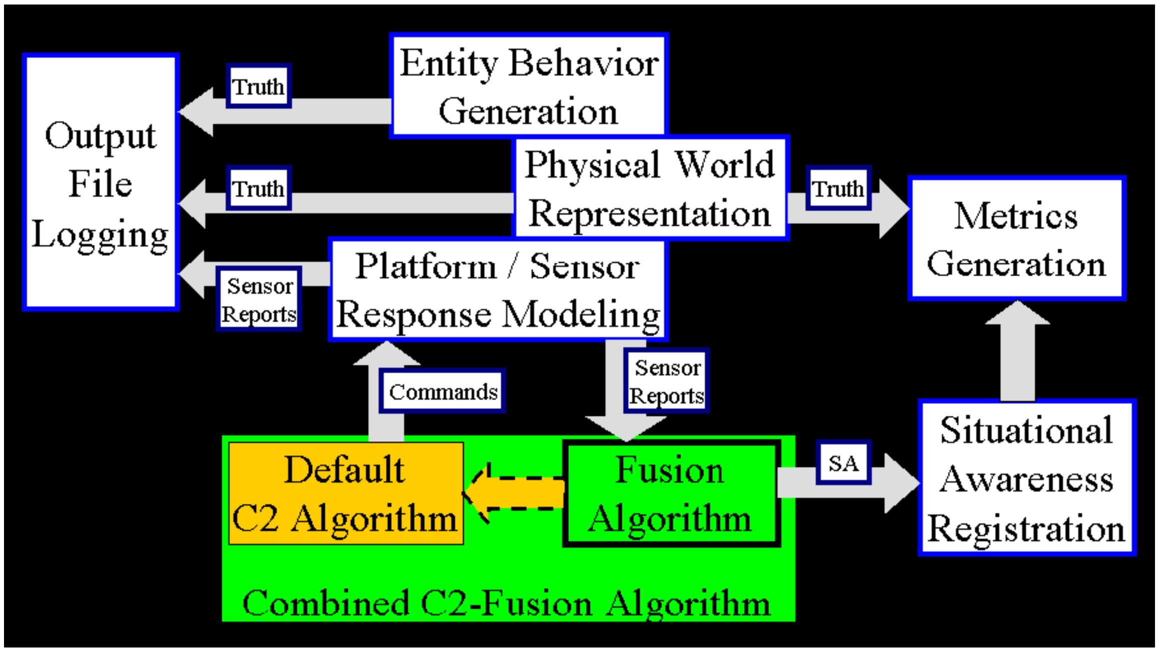

Entity Behavior Generation
Output File Logging
Truth
Truth
Physical World Representation
Truth
Metrics Generation
Sensor Reports
Platform / Sensor Response Modeling
Commands
Sensor Reports
Default C2 Algorithm
Fusion Algorithm
SA
Situational Awareness Registration
Combined C2-Fusion Algorithm

Figure 12: The components of the Northrop Grumman's cognitive test bed

The test bed supports two modes of internal algorithms, as shown in Figure 13. A default command and control (C2) module provides a simplistic path following behavior for the platforms to generate sensor reports for testing of a sensor fusion-only algorithm. In addition, the default C2 behavior can be removed, and a combined C2-fusion algorithm can be used to not only generate the situational awareness, but also drive the positioning of the various UAV assets.

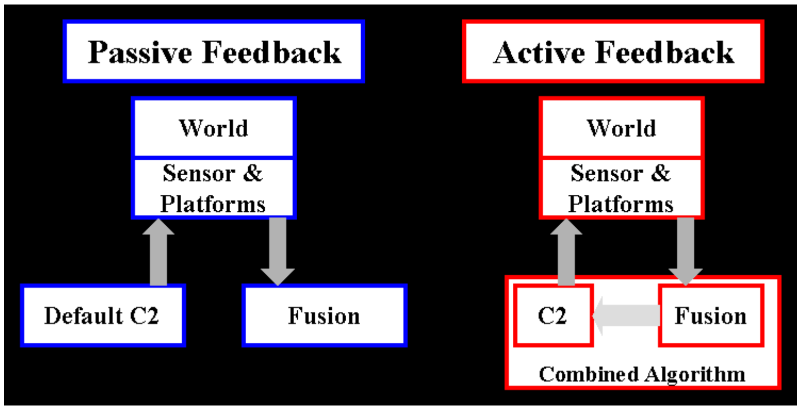


Figure 13: The test bed provides a capability to test both algorithms which actively drive the UAV platforms and sensors and those that only process the reports as generated.

The test bed models a world specified by a map and a region of interest. Sensors reside on platforms, which move over a two-dimensional region of interest. Within this region of interest, entities with various movements and behaviors may reside. A sensor passing within range of an entity generates a report. The report includes estimates of the location and velocity of the detected entity; and it includes one reading of several possible components that make up the entity's signature. A single report may not uniquely identify an entity; but, an aggregate of reports provides a better picture of the total signature, increasing the likelihood of identification of the type of entity observed.

Each entity type has a signature that is defined by some number of signature (feature) elements. Each feature element in the signature can be represented as a probability distribution function within a separable (but not necessarily independent) dimension in an *N*-dimensional space. For simplicity, the test bed assumes a Gaussian probability distribution function, represented by a mean and a standard deviation.

$$\vec{S}_T = (S_{1T}, S_{2T}, S_{3T}, S_{4T}, \ldots, S_{NT}) \qquad (1)$$

$$\vec{S}_{T\sigma} = (S_{1T\sigma}, S_{2T\sigma}, S_{3T\sigma}, S_{4T\sigma}, \ldots, S_{NT\sigma}) \qquad (2)$$

where S_T is the mean signature of the T^{th} entity type;

$S_{1T}, S_{2T}, \ldots S_{NT}$ are the *N*-dimensional mean signature (feature) elements of S_T; and

$S_{T\sigma}$ similarly represents the standard deviation component.

From these values, the probability distribution function (PDF) can be calculated by the following equation:

$$PDF_T(\vec{v}) = \prod_{n=1}^{N} \frac{e^{-\frac{(v_n - S_{nT})}{2 \cdot S_{nT\sigma}^2}}}{\sqrt{2\pi} S_{nT\sigma}} \tag{3}$$

where : S_{nT}, $S_{nT\sigma}$ are given as per equations 1 and 2; and PDF(v) is the PDF over the dimensions given by the vector v .Note that this PDF gives the probability that a specific type *T* will generate a signature at the *N*-dimensional signature location given by the vector *v*.

Using a priori knowledge of these probability density functions, sensor reports can be processed with a number of algorithms to determine the classification of an entity.

- **Inputs to the test bed**:

The user interacts with the test bed by modifying two XML input files:

1. dstb.xml: In this xml file , general parameters associated with the configuration of the world being simulated are specified. These parameters include types of entities, platforms, sensors, signatures, entity behaviors, time for which the world needs to be simulated, periodic sensor reporting interval etc.
2. scenario.xml: In this xml file, scenario dependent parameters, such as the map of the region, size of the region of interest and the number of platforms, sensors and entities are specified.

These two files support a wide variety of flexibility in configuration. Additional command line flags may also be added to control the behavior of the test bed.

- Outputs of the test bed :

1. The test bed, written completely in C++, provides three developer APIs :

    - The Sensor Report API allows a fusion algorithm to get sensor report and truth report information at discrete intervals of simulation time.
    - The C2 API allows a Command and Control algorithm to command Platforms
    - The SA (situational awareness) API allows a fusion algorithm to register its awareness.

When testing a fusion algorithm while using the default command and control algorithm, the fusion component uses the sensor report API to obtain sensor and truth reports every discrete interval of simulation time, processes the sensor reports and compares it to the provided truth information to determine the level of functional correctness.

2. The sensor and truth report data over all time steps of the simulation are dumped into an xml file at the end of a simulation. This file provides an efficient means of testing fusion algorithms outside the test-bed simulation framework.

A sensor report consists of the following: a value in one (or more) signature dimensions, a location and error ellipse, a velocity vector and error ellipse, time with error, the identification of the sensor generating the report . A truth report consists of the following: entity id, entity type, track id, true location, true velocity, true time

In addition to this, the test bed can also be configured to generate a track id for each sensor report. This track id corresponds to internally generated tracks created based on the Occam Razor's principle [61]. Creation of internal tracks can be optionally turned on/off based on whether the fusion component creates tracks as well as performs data association or performs data association alone (like in the case of IDA).

4.2.3 Application of IDA to the Northrop Grumman test bed

The original MATLAB code of the Information Matrix Data Association (IDA) algorithm was provided by the algorithm developers at Stanford University. This code was converted to C and applied to the data generated by the Northrop Grumman test bed simulations.

In addition to the truth and sensor report data, the IDA algorithm requires a means of correlating signature values pertaining to different signature dimensions to entity types modeled by the test bed. The test bed supports 50 types of entities each defined by a combination of 10 signature feature dimensions. Since signatures are assumed to be represented by a normal distribution in the test bed, they are characterized by a mean and standard deviation as described in the previous section. The mean and standard deviation of signature values in each signature dimension corresponding to various entity types is extracted from the test bed in order to facilitate correlation of signature values to unique entity types by the IDA algorithm.

The IDA C code has been integrated into the test bed in the fusion component. Each time, the test bed is run with the option for automatic generation of internal tracks .This ensures that each sensor report has an associated track id and this track id is used to select the column of the information matrix that needs to be updated corresponding to an incoming sensor report .Every discrete time step interval, a chunk of sensor reports and truth reports is delivered to the fusion component as a part of the simulation.

Each time step, the following operations (explained in detail in section 3.2.1) are performed by the IDA algorithm using the sensor data:

1. Mixing Updates:

From the location data obtained from the sensor reports, the number of pair wise mixing updates is determined based on a predefined cartesian distance threshold. Subsequently the Information Matrix data structure is updated as per the rules mentioned in section 4.2.1

2. Observation Updates:

Each sensor report is classified and the probabilities of it corresponding to each of the 50 different entity types is determined. This information along with the track id information in the sensor report is used to update the Information Matrix as per the rules described in section 4.2.1

3. Inference:

The data association of the previous time step (maintained in a single dimensional array of size N) and the updated information matrix for the current time step are used as inputs for the inference operation wherein based on the state of the current Information Matrix, using simulated annealing an attempt is made to determine the track-entity id associations for the current time step.

4. **Measuring accuracy:**

The data associations are maintained in a single dimensional array of size N, where the index of the array pertains to the track id and the value of the element pertains to the inferred entity id corresponding to the track for the particular time step. At the end of the inference step, an updated array of size N is obtained corresponding to the set of data associations in the current time step.

E.g. If the data association array is as shown in Figure 14, it means that track 1 is associated with entity id 1 at the end of the inference step, track 2 with entity id 3, track 3 with entity id 2 and so on. To determine the accuracy of associations, the truth reports corresponding to the internally generated tracks are used. For example if the entity id in the truth report for track 1 is 1, it corresponds to a correct association for track 1 since the association obtained from the inference step is also the same. The total number of correct associations is determined through this correlation with truth reports every time step and a score is maintained. At the end of the simulation, the individual correct association scores pertaining to time steps is summed up and divided by the total number of time step intervals simulated to determine the average tracking accuracy value.

1
2
4
7
6
5
8
10
9
3
11
12

Figure 14: Example data association array obtained at the end of the inference step . Here the index into the array represents the track id, the value of the array element represents the entity id associated with the track at the end of the inference step.

Since we are concerned only with the fusion component of the test bed, instead of running IDA as a part of the simulated test bed framework, it is much more efficient to generate datasets from the test bed using different input configuration files and simulation windows and then using these datasets as inputs to the IDA algorithm, running natively on a machine. The datasets generated by the test bed simulations comprise time stamped sensor and truth reports in xml format. These xml files are converted to text files and used as input to the IDA algorithm. In the rest of the thesis, whenever we refer to the NG test bed data, we would be referring to these datasets generated from test bed simulations with different input configuration files.

5. Entity Behaviors

The Northrop Grumman test bed provides a parameterized framework to vary the behaviors of entity types by changing parameters like the start and end time of the entity's movement, the start and end locations, the route taken, commute and pause time, maximum speed of the entity type etc. Changing the behaviors of entities in a data set affects the computation time spent in the Mixing Update and Observation Update sub-kernels of the IDA algorithm. This is due to the fact that change in behaviors result in change in the number and type of mixing events as well as sensor reports. In our study, we use data sets with two different sets of behaviors associated with the entity types. In addition to that, we vary the number of entities in the configuration files to generate new data sets. Varying the number of entities scales the problem size in terms of the size of the primary data structure (Information Matrix) involved in the computation. Also it changes the amount of mixing and the type and number of sensor reports delivered to the fusion component every time step.

4.3 Real time problem formulation

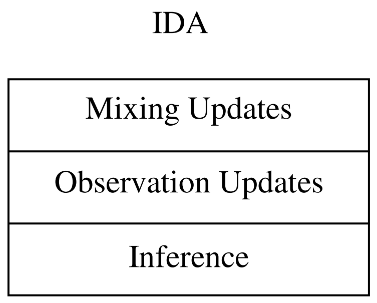


Figure 15: Computations to be performed on new sensor data every time step interval

Let ‘tint’ be the time step interval size/periodic interval at which sensor data is delivered to the fusion algorithm.

Let ‘trefresh’ be the periodic interval at which the user expects tracking accuracy updates

Figure 15 shows the computations that need to be performed on new sensor data every time step interval. The following points outline the salient features of the real problem formulation:

- System objective: Maximize tracking accuracy value delivered to the user every ‘trefresh’ seconds.
- Task model: Three sets of periodic , dependent tasks initiated every period/time step interval given by ‘tint’
- Deadline: To meet the system objective in the best possible way, each of the three tasks needs to be completed before the arrival of the next set of sensor reports. So deadline is ‘tint’ seconds after the arrival of the current set of sensor reports.

- Nature of deadline: Soft because no catastrophic failures result from missing a deadline. However utility of the system decreases as more and more deadlines are missed.

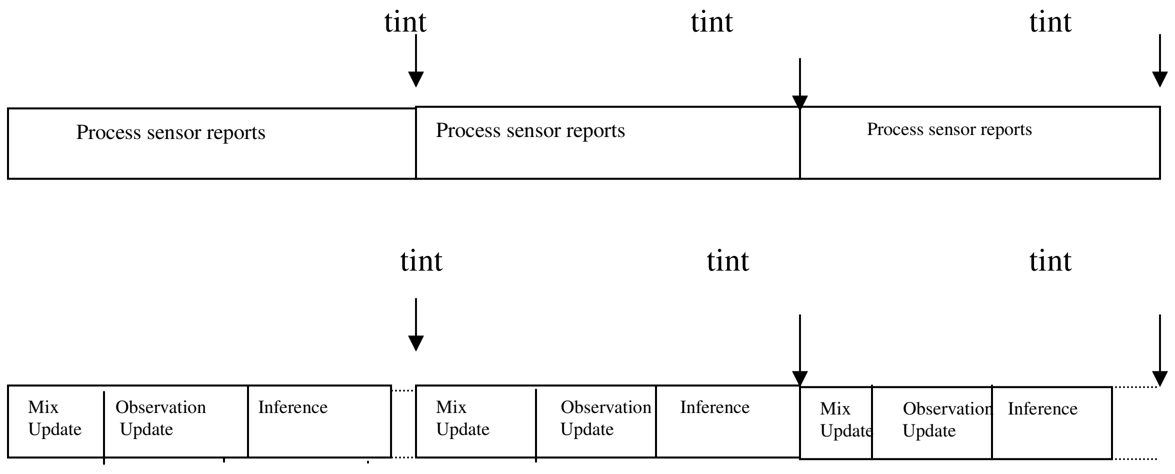


Figure 16: Ideal case where the computations on each set of sensor reports is always completed before the next set is delivered

Figure 16 shows the ideal case scenario in which the computations to be performed on each set of sensor reports is completed before the next set arrives. However due to the dynamics of the environment in which the system works and the data dependent nature of the three tasks that need to be performed each time step, there is a high variance in the amount of time taken to complete these tasks from time step to time step.

Let us consider a single dataset for 3000 entities with the default set of behaviors in the test bed and understand the variance in computation time of each of the tasks across time steps. Figure 17 shows how the computation time spent in each of the three tasks varies across time step intervals. The value of 'tint' in this case is 1second i.e. a new set of sensor reports is delivered every second from the simulations. The timing measurements pertain to single threaded performance on a 3GHz dual processor Xeon machine.

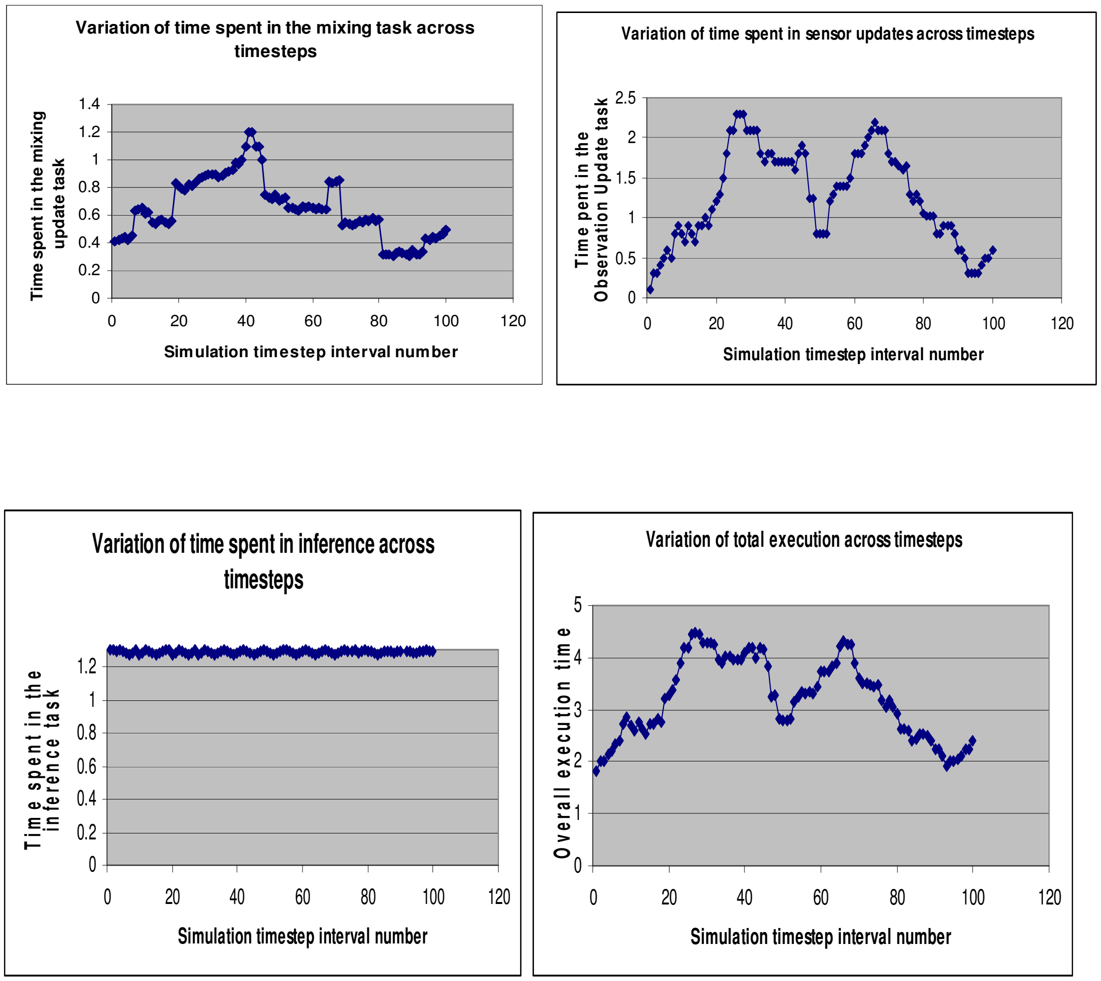


Figure 17 Variation in computation time spent in each of the three tasks sacross time step intervals.

From Figure 17, the following observations are made:

- The computation time to perform all the three tasks varies from 1.8 secs to 4.5 seconds.
- There is a high variance in the amount of time spent in the mixing task across time steps. This is because this is dependent on the number of pair wise mixing

events, which is dependent on the dynamics of the world being simulated and the behaviors of the entities being tracked.

- There is also a significant variance in the amount of time spent in the observation update task across time steps. This is due to the fact that the number of sensor reports delivered for processing varies from time step to time step.
- The variance in the amount of time spent on the inference component is not very high. This is because the number of iterations of simulated annealing is fixed and the work done in each iteration is approximately proportional to the number of entities being tracked which determines the size of the Information Matrix, the main data structure involved in this computation. The slight variations we see in the graph are mainly due to measurement differences and do not reflect variations in computation time.

The reasons why conventional real time systems design/scheduling techniques are unable to handle such applications efficiently are:

- As mentioned earlier, static design methodologies would attempt to determine

Worst Case Execution Times (WCET) for each of the tasks and design a system with enough processing power/resources to meet the deadlines. Since the computations associated with the tasks are input data dependent, in almost all cases, the WCET would be unbounded. Even if we assume that the WCET is bounded and determinable and design a system to meet WCET requirements using static scheduling policies, the resulting system would be underutilized during most phases of operation.

- Use of dynamic scheduling policies like Earliest Deadline First might be more amenable to be applied to such an application, but again if designed for handling less than the Worst Case Execution Time(WCET) scenarios, missed deadlines and resulting domino effects would cause unpredictable dropping of tasks and hence adversely affect the tracking accuracy updates provided to the human user every ‘trefresh’ time.

Given the dynamics of the problem scenario, we propose a methodology of exploiting certain application features to ensure that the system objective is met with sufficient solution quality when a deadline arrives rather than supporting and designing for the worst-case execution of the program.

In the next two chapters, we describe in detail how certain properties of the three types of tasks in our reference application (IDA) allow us to make performance accuracy trade-offs at run time, meet deadlines and meet the system objective in the best possible way given resource and time constraints.

Chapter 5

5. Soft computations and their characterization

As mentioned in section 1.4, a growing number of kernels [47, 48, 49] currently offer and several others can be structured to offer [50, 51, 52] a simple means of trading off computation time for the quality of results. We call such kernels "soft kernels" or "soft computations." The chief characteristic of these computations is the fact that they are able to provide cruder results before they complete, or they may execute for a long time refining an already adequate result. In other words, such computations may be able to provide useful, incremental results before completing execution and also ensure graceful degradation of the quality of results as the computation time decreases. By providing opportunities for obtaining "anytime" solutions at deadlines, these computations allow us to meet a real time system objective with sufficient quality without having to support worst case program execution.

We demonstrate the benefits of exploiting soft computing properties of kernels in the context of our system prototype described in Chapter 4. Two main design issues involved in trying to build a system where soft computations are exploited to meet deadlines with acceptable solution qualities are:

1. Quantifying the gain in solution quality of the application as a function of time spent in soft computations.
2. Determining the optimal composition of "tasks" given a specific amount of resources/ time to meet a deadline.

We address these design issues in this chapter. More specifically, we identify the presence/absence of soft computing properties in the tasks/software components of our reference application and characterize the soft computations using performance profiles. We then use these performance profiles to build models to determine the optimal composition of a group of tasks given a specific resource and timing constraint to meet the system objective in the best possible way.

5.1 Properties of soft computations

The desirable properties of computations for them to be classified as "soft computations" or "soft kernels" are as follows:

1. Measurability: The quality of the system output should be measurable and the computation should have a measurable impact on the system objective or the quality of the system output. For example in the case of our system prototype, the system output is measured as the tracking accuracy, measured with respect to the ground truth, which is a measurable system objective. Measurability is important because without being able to objectively determine the impact of computations on the system solution quality, it is difficult to characterize them and understand computation time vs. solution quality trade-off opportunities that they might offer.
2. Monotonicity: The impact of the computation on the system objective/quality of the system output should be a non-decreasing function of time. If a computation is non-monotonic, it cannot be guaranteed that spending more time in the computation would yield a better solution quality. Essentially it means that the computation does not offer a trade-off between computation time and system solution quality.

3. Consistency: The impact of the computation on the system objective/quality of the system output should be correlated with computation time. In general, algorithms do not guarantee a deterministic output quality for a given amount of time, but it is important to have a narrow variance so that quality prediction can be performed.

4. Diminishing returns: The improvement in solution quality should be larger at the early stages of the computation, and it should diminish over time. It is desirable for a computation to provide diminishing returns for it to be characterized as a soft computation because the “diminishing returns” characteristic ensures graceful degradation of system solution quality as time allocated to the computation is decreased.

5. Recognizability: This pertains to the ability to determine the impact of a computation on the quality of the system solution at run time. For example, when solving a combinatorial optimization problem (such as path planning), the quality of a result depends on how close it is to the optimal answer. In such a case, quality can be measurable but not recognizable. Similarly in the case of our system prototype, the quality of a data association solution depends on how close the associations are to the ground truth. Hence again though the quality is measurable, it is not recognizable. Recognizability becomes extremely important if the quality of the system solution as a function of time spent in soft computations needs to be monitored at run time for the purpose of adapting and learning performance profiles online. We discuss what performance profiles are in detail in the next section.

We initially look for the first four desirable characteristics of soft computations in the components of our system prototype (discussed in detail in section 5.3). We study how much benefit soft computations that do not possess the "recognizability" characteristic, can provide for real time performance. Subsequently, we perform a sensitivity study to determine how much more benefit can be accrued if the computation is "recognizable" and thus facilitates online learning and adaptation of performance profiles, that characterize soft computations.

5.2 Performance profiles:

Soft computations are characterized using performance profiles. Performance profiles quantitatively summarize the improvement in the quality of output as a function of the time spent in the computation. A performance profile of a soft computation can be represented as Q(t) where Q(t) represents the quality of the output when time 't' is spent in executing the soft computation.

Performance profiles are typically constructed empirically by collecting statistics on the performance of an algorithm. The raw data that are collected specify the particular quality of results as a function of computation time.

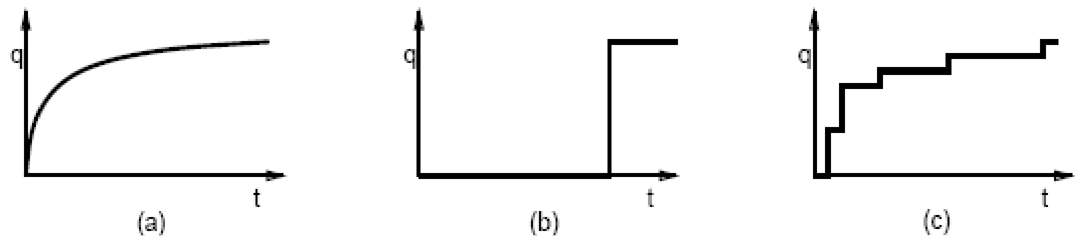


Figure 18: Typical performance profiles. Figure 18b shows the performance profile for a traditional algorithm. Figures 18a and 18c show performance profiles for soft computations

Figures 18b is the performance profile for a traditional algorithm, the x axis pertains to time spent in the computation and the y axis pertains to the corresponding quality of solution that can be expected. What this curve essentially illustrates is the fact that a traditional algorithm has just one output /solution which can be obtained provided a sufficient amount of time is provided to the task for execution. If sufficient amount of time is not allocated to the algorithm, no output/solution is obtained. Figures 18a and 18c are examples of performance profiles for soft computations. What these figures illustrate is that when using soft computing kernels, output/solutions of lesser quality/value are obtainable even if execution of the kernels are restricted to a fraction of the time required for the computation to deliver the best possible value/solution to the problem. As evident from Figures 18a and 18c, these two performance profiles illustrate "measurability," "monotonicity" and "graceful degradation" for the computation they represent. Whether the computation possesses the "consistency" property can be evaluated only when the performance profile is used for quality prediction while the computation is being executed and the variance between the predicted quality and the actual quality obtained from the algorithm execution is determined.

5.2.1 Quality metrics for characterizing soft computations:

In order to draw performance profiles, objective metrics for measuring the quality of the output must be defined. Such quality measures specify the difference between the approximate result and the exact result. They are "objective" in the sense that they are a property of the algorithm itself, independent of its possible applications. From a pragmatic point of view, it may seem useful to define a single type of quality measure to be applied to all algorithms having soft computing properties. However quality measures must match the nature of the algorithm they describe. Soft computations are unique and every algorithm may have a different objective quality measure. However the following three metrics have been found useful and relevant in the context of many "soft" kernels [44]:

1. **Certainty:** This metric reflects the degree of certainty that a result is correct. The degree of certainty can be expressed using probabilities, fuzzy set membership, or any other method of expressing uncertainty. For example, consider a "soft" diagnosis algorithm that is based on combining more and more evidence as computation time increases. The certainty that the diagnosis is correct increases as a function of run-time. With this type of "soft" kernel, there is always a possibility that the correct results are completely different from the ones generated by the algorithm.
2. **Accuracy**: This metric reflects how close the approximate result is to the exact answer.
3. **Specificity:** This metric reflects the level of detail of the result. In this case, the "soft" algorithm always produces correct results, but the level of detail is increased over time. For example, consider a hierarchical planning algorithm that first returns a high level abstract plan. Each step in the abstract plan is a "macro" step that needs to be refined by further planning. As computation time increases, the level of detail is increased until the plan is composed of base-level steps only that can be easily followed.

5.2.2 Building performance profiles for individual soft kernels:

Finding the performance profile of an individual /elementary soft kernel can be difficult and may require an extensive computation effort especially when it is based on simulation of the algorithm. In some cases, such as numerical analysis algorithms, the performance profile can be derived by direct analysis of the algorithm, but the general case is more complicated. For example, in many iterative algorithms, such as Newton's method, the error in the result is bounded by a function that depends on the number of iterations. In such cases, the performance profile can be calculated once the run-time of a single iteration is determined. In general, however, such structural analysis of the code is hard because the improvement in quality in each iteration and its run-time may be unpredictable. A more general method is based on gathering statistics on the performance of the algorithm in many representative cases. Statistical performance profiles are the easiest to construct but take the most storage space and the longest amount of time to instantiate. A statistical profile uses a large number of samples to create a database of (computation time, output quality) entries. With this information the database can then be used to make predictions about the expected quality given time and input parameter information.

5.2.3 Representation of performance profiles of individual soft computations:

Performance profiles of individual soft computations can be represented either by a closed formula or as a table of discrete entries.

Since performance profiles are normally monotone functions of time, they can be approximated using a certain family of functions. Once the quality map is known, the performance information can be derived by various curve fitting techniques[49] .

The discrete representation of performance profiles is based on a table that specifies the quality values for certain possible time allocations. The size of the table is a system parameter that controls the accuracy of performance information.

We discuss in detail how soft computations are characterized using performance profiles in the context of our reference real time scenario implementation in the next section.

5.3 Identification and characterization of individual soft kernels in the reference application

We performed experiments to determine if the three tasks/ sub-kernels (Mixing Update, Observation Update and Inference) that form a part of our reference implementation possessed one or more of the properties mentioned in section 5.1 that characterize "soft" computations. We observed that all three sub-kernels satisfy the "quality measurability" property that is needed to characterize soft computations since the impact of time spent in each task on the accuracy of the application can be measured. The Observation Update and Inference sub-kernels /tasks also satisfy the other three desirable properties of soft computations and hence are categorized as "soft" kernels. The Mixing Update sub-kernel updates the information matrix data structure based on the spatial state of entities in the scenario. It causes degradation in the state of the information matrix data structure based on the number of tracks that mix in the scenario in a particular time. This computation ensures that the state of the world is appropriately reflected in the state of the Information Matrix. As a result, it is important to ensure that the Mixing sub-kernel always runs to completion. Since the Mixing Update sub-kernel causes degradation in state of the Information Matrix, the impact of the Mixing Update sub-kernel on the system solution quality is non-monotonic, inconsistent and does not ensure graceful degradation of solution quality as time spent in the Mixing Update sub-kernel is decreased. So we conclude that since the Mixing Update sub-kernel does not satisfy the three main desired properties of soft computations, it should be categorized as a "non-soft"/"hard"/"mandatory" kernel which always needs to be executed to completion. We provide details of the methodology adopted by us for characterizing the "soft kernels" in the following sections.

5.3.1 Observation Update sub-kernel /task

This sub-kernel performs updates of the information matrix of the IDA sensor fusion algorithm using local sensor report observations. Each incoming sensor report is first classified and the probability values so obtained are used to perform the update of the Information Matrix as described in Chapter 4. The amount of time spent in this task varies across time steps based on the number of incoming sensor reports. Another observation is that some sensor reports do not add much additional value to the final outcome/accuracy due the fact that either they are redundant (more than one sensor report per entity) or are not critical in the context of this specific application scenario. This notion of criticality will become clearer when we discuss the experiments we performed in the next few paragraphs.

Our observations indicated that it might be possible to trade-off computation time for accuracy in this sub kernel by performing critical sensor updates pertaining to the domain instead of performing all updates every time step. In order to determine whether this sub kernel possesses the desirable "soft computing" properties, the following experiments were performed:

- **Experiment 1:**

We first varied the amount of time spent in the Observation Update sub-kernel and studied its impact on the accuracy of the application. We did this by varying the percentage of sensor updates applied in each time step from 0% to 100% in steps of 5% and then determined the average accuracy value across the different time steps. In all cases, we executed the Mixing Update and Inference sub-kernels to completion. Figure 19 shows the variation of tracking accuracy as a function of time spent in sensor updates. This data pertains to the data set with 3000 entities and default behaviors. The tracking accuracy and execution time pertain to the average accuracy and execution time measured across all the 500 simulation time steps. As evident from the graph in Figure 19, we observed that the observation update computation is monotonic. We also realized that the order in which the sensor updates are applied is important. This was based on the observation that some updates applied could be more critical than others while other updates may be redundant due to the delivery of more than one sensor report per entity.

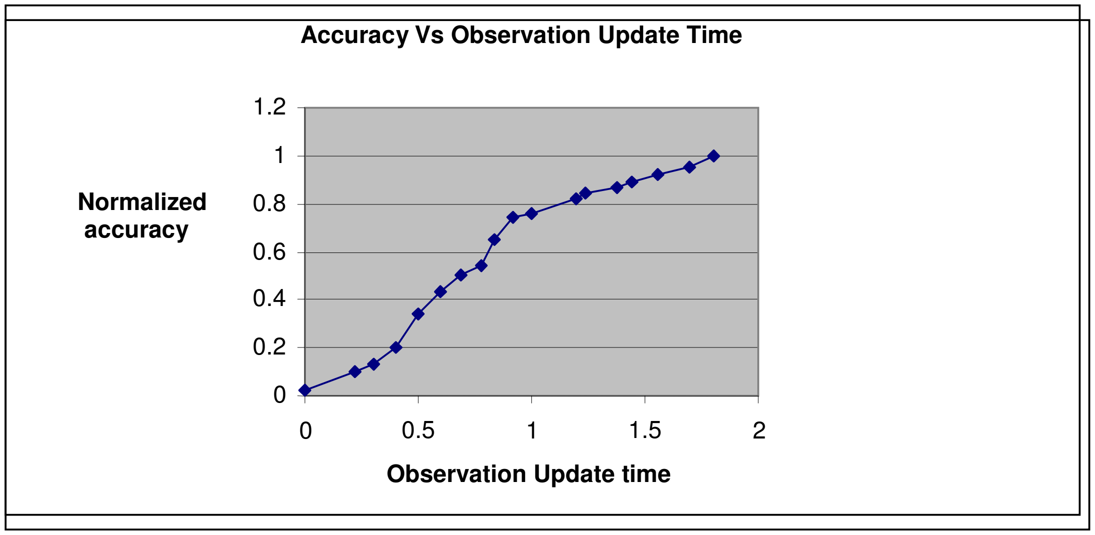


Figure 19: Variation in accuracy as amount of time spent in the ObservationUpdate task is changed

- **Experiment 2:**

Based on the intuition that determination of an order of applying updates based on sensor report criticality might help improve the consistency of the soft computation and also ensure more graceful degradation with decrease in computation time, we divided the sensor reports into 4 groups:

1. **Sensor reports pertaining to mixing events:** This group comprises all the sensor reports pertaining to tracks that were involved in mixing. It is important to note here that there could be more than one sensor report pertaining to a track involved in mixing. All these sensor reports are included in this group.

2. Sensor reports pertaining to similar entities: This group comprises all sensor reports pertaining to similar entity types. How similarity of entity types is determined is discussed later in this section.

3. First time sensor reports: After filtering out the above two categories of sensor reports, the reports pertaining to tracks that have not received a previous sensor update in the time step are included in this group

4. Redundant sensor reports: All the remaining reports obtained after filtering out reports pertaining to the above three groups are put in this category.

The grouping of sensor reports was based on the following observations:

1. Sensor reports pertaining to mixing events are critical owing to the uncertainty introduced by the mixing of tracks.

2. Sensor reports pertaining to similar types of entities are critical owing to the fact that without sufficient observation updates; it might become difficult to distinguish between two entities having similar signature values. The similarity between entity types is determined offline using the Mahalabonis distance metric. This is a very popular similarity/proximity metric used in statistical data mining techniques specifically in the arena of cluster analysis [61].

- **Heuristic for assigning priorities to sensor updates:**

The following steps are followed to facilitate category based ordered updates of sensor reports:

1. To each sensor report information data structure, one additional unsigned integer element is added which contains the category value. Also an additional array of size N is maintained which contains two bits of information a. whether a particular track is involved in mixing and b. whether a sensor report pertaining to that track has already been processed.

2. When the mixing updates are performed, the corresponding category related

sensor report element is assigned a value of 3. Also the corresponding mixing bit entry in the array of size N is set to 1.

3. As each report is classified, if the category is not set and corresponding value of the N dimensional array is set, the category element is set to a value of 3. Otherwise, based on the classification result, if the type of entity belongs to one of the predetermined similar types, the category value is set to 2 else if the bit in the N dimensional array for first time reports is not set, it is set to 1 and the category element is assigned a value of 1. The sensor reports are then sorted in descending order of category values.

Using the above heuristic and category based ordering of sensor updates, we obtained a performance profile for the Observation Update task for coarse granularity sensor updates based on categories. This was done in the following manner.

1. The mixing update and inference tasks were performed every time step.
2. We obtained 4 data points by determining average accuracy values by performing updates for sensor reports pertaining to mixing events alone; mixing events and similar entities; mixing events ,similar entities and first time reports; all reports.

In order to make sure that the order of updates based on the above mentioned order of categories is the best, the above curves were obtained for all possible permutations of the category based orderings. Figure 20 shows the average computation time vs normalized accuracy graphs for these 6 set of experiments.

As evident from Figure 20, the following ordering of category based sensor updates provides, the best performance profiles for the Observation Update task .

-Updates pertaining to mixing entities

-Updates pertaining to similar entities

-First time sensor updates

-Redundant sensor reports

This was observed for all the data sets under consideration. Hence we adopted this heuristic to drive the order in which sensor updates were applied to the Information Matrix:

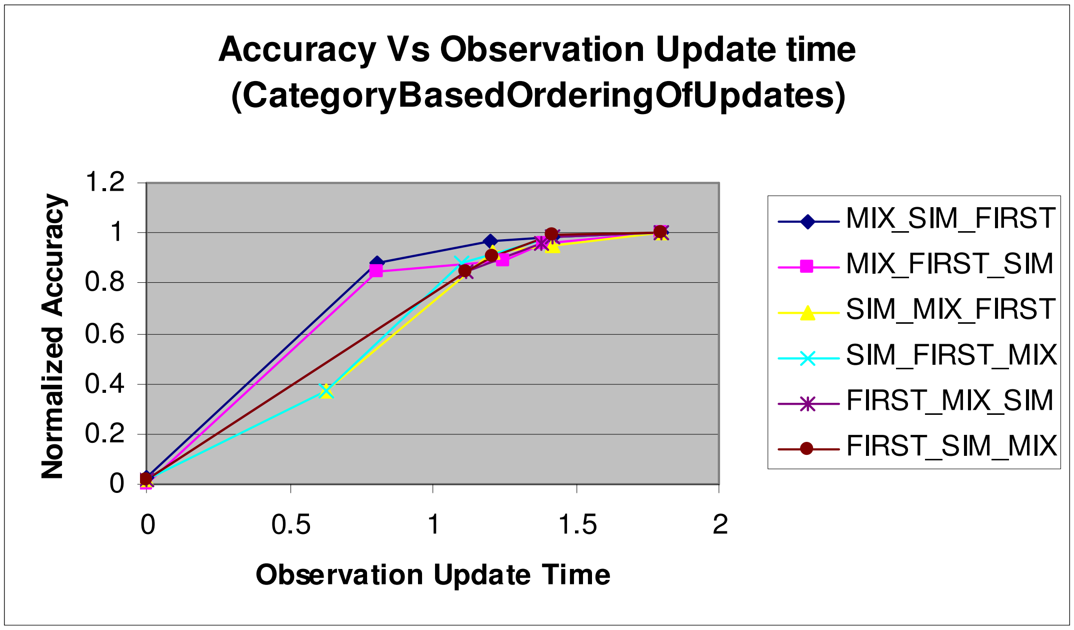


Figure 20: Variation of accuracy as a function of time spent in the observation subtask when applying category based ordered updates

We thereafter obtained performance profiles for a finer granularity of sensor updates, more specifically; we obtained accuracy values by performing experiments by varying the granularity (number of sensor reports) of the Observation Update task using the following approach

- Apply 5-100% mixing sensor updates each time step in steps of 5%

- Then apply 100% mixing sensor updates + (5-100%) sensor updates pertaining to similar entities in steps of 5%
- Then apply 100% mixing sensor updates + 100% sensor updates pertaining to similar entities + (5-100%) first time sensor updates in steps of 5%
- Then apply all updates.

Figure 21 shows the performance profile for the Observation Update so obtained. The results are pertaining to the same dataset with 3000 entities and default behaviors.

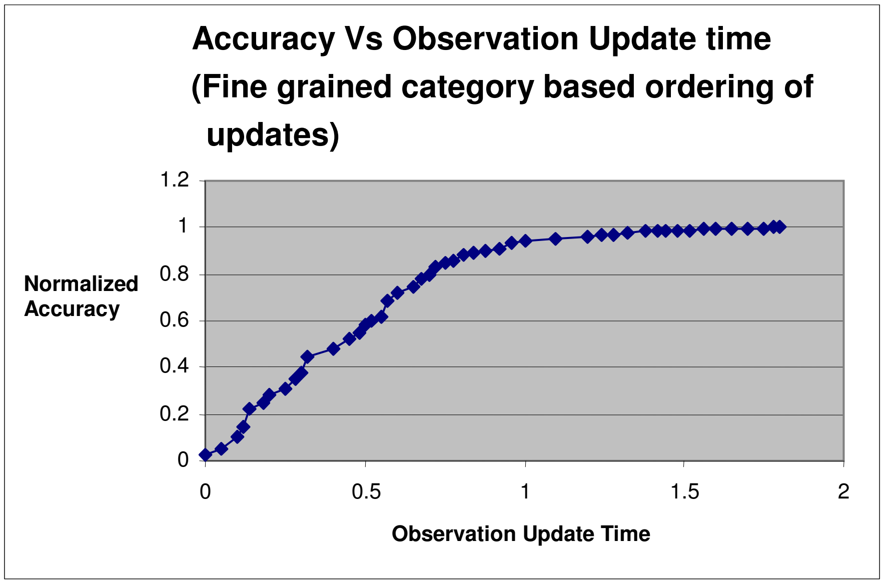


Figure 21: Variation of accuracy as a function of time spent in the observation subtask when applying category based ordered updates at a finer granularity

Based on the experiments performed by us pertaining to the Observation Update sub kernel, we observed that this kernel possessed all the four desirable properties for soft computations. Hence this task/sub-kernel was categorized as “soft” in the context of our reference real time scenario.

5.3.2 Inference sub-kernel /task

This sub-kernel attempts to determine the best data association of tracks with entity ids in a time step by performing inference using a simulated annealing formulation.

Simulated annealing is a technique that has been used in various combinatorial optimization problems. It is a technique to find a good solution to an optimization problem by trying random variations of the current solution. The slower the cooling schedule, or rate of decrease of the initial specified temperature for the annealing schedule, the more likely the algorithm is to find an optimal or near-optimal solution. The chance of getting a good solution can be traded off with computation time by slowing down the cooling schedule. The slower the cooling, the higher the chances of finding the optimum solution, but the longer the run time. By varying the number of iterations of Simulated Annealing, the rate of cooling can be changed and hence accuracy can be traded off for computation time. The original inference kernel worked with a fixed number of inference iterations. We try to exploit the potential for trading off computation time for accuracy by varying the number of inference iterations and hence varying the cooling schedule of the simulated annealing formulation. Figure 22 shows the basic formulation of a Simulated Annealing algorithm. The inputs to the formulation are an initial solution (in our case, data associations of the previous time step) and a cooling schedule specified by a start temperature (T), a stop temperature (T') and the number of iterations (N). At each iteration, the temperature is decreased to T/N. Essentially this means that by varying N, we can change the cooling schedule and thus exploit a trade-off between the run time of the inference step and the solution quality. In addition to this, the solution that is obtained for evaluation or assessment each iteration is based on the current state of the information matrix (in the case of our system prototype) and is not randomly chosen. This ensures that we progressively obtain better solutions (data associations) as we perform more iterations. However the major changes in associations across time steps are identified in the first few iterations of the simulated annealing inference and hence after some number of iterations, for most time step intervals, the accuracy levels off.

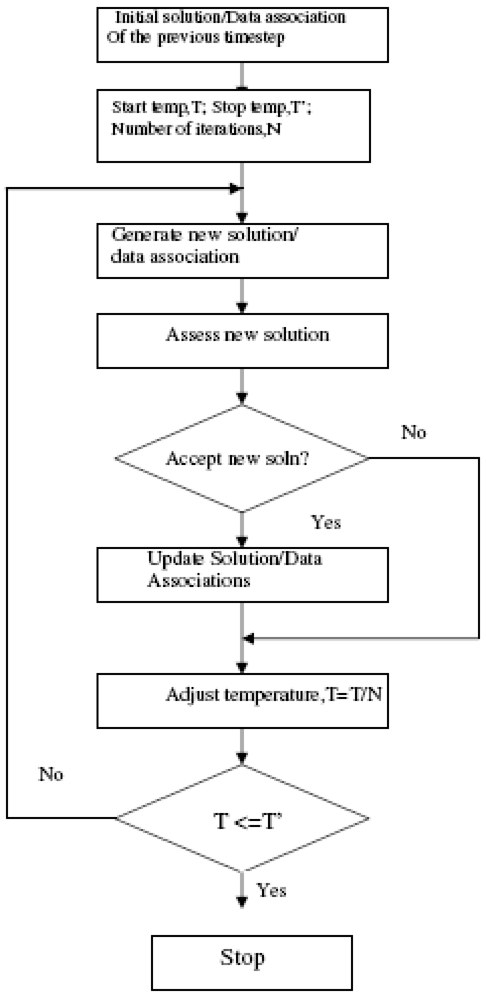

Initial solution/Data association Of the previous timestep
Start temp,T; Stop temp,T'; Number of iterations,N
Generate new solution/ data association
Assess new solution
Accept new soln?
No
Yes
Update Solution/Data Associations
Adjust temperature,T=T/N
No
T <=T'
Yes
Stop

Figure 22: Basic structure of a simulated annealing formulation

In order to determine whether the inference sub-kernel possesses the desirable soft computing properties, we obtain average accuracy values for the application by performing the Observation Update and Mix Update sub-kernels to completion every time step and by varying the number of iterations of the simulated annealing formulation between 1 and MAX [MAX=maximum number of iterations originally used in the algorithm. The value of max iterations in the original algorithm is 30]. Figure 23 shows how accuracy of the system varies with time spent in the inference sub-kernel. The data in this figure corresponds to the dataset with 3000 entities and default behaviors. As evident from the figure, this sub-kernel possesses the soft computing properties of monotonicity and graceful degradation and hence is categorized as a "soft" kernel.

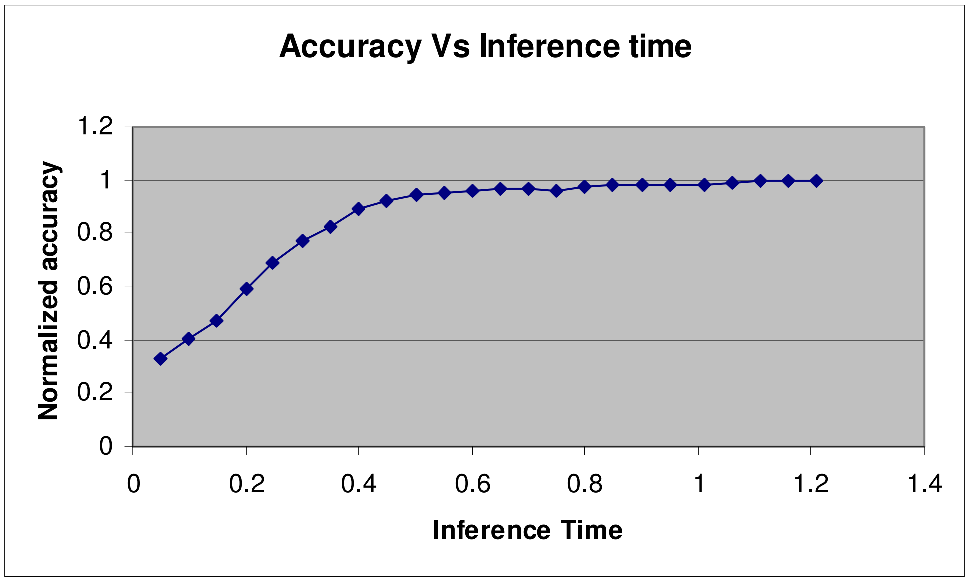


Figure 23: Performance profile of the inference task

5.4 Composing systems using performance profiles:

Given a system composed of a combination of “hard” and “soft” kernels/tasks as in the case of our reference implementation (where the Mixing Update task has been identified as “hard” while the Observation Update and Inference tasks have been identified as “soft”) and given a specific program structure, it is important to determine the optimal allocation of time to the components for any given total time allocation so as to meet the system objective. This problem of optimal composition of such tasks is an NP complete problem [52].

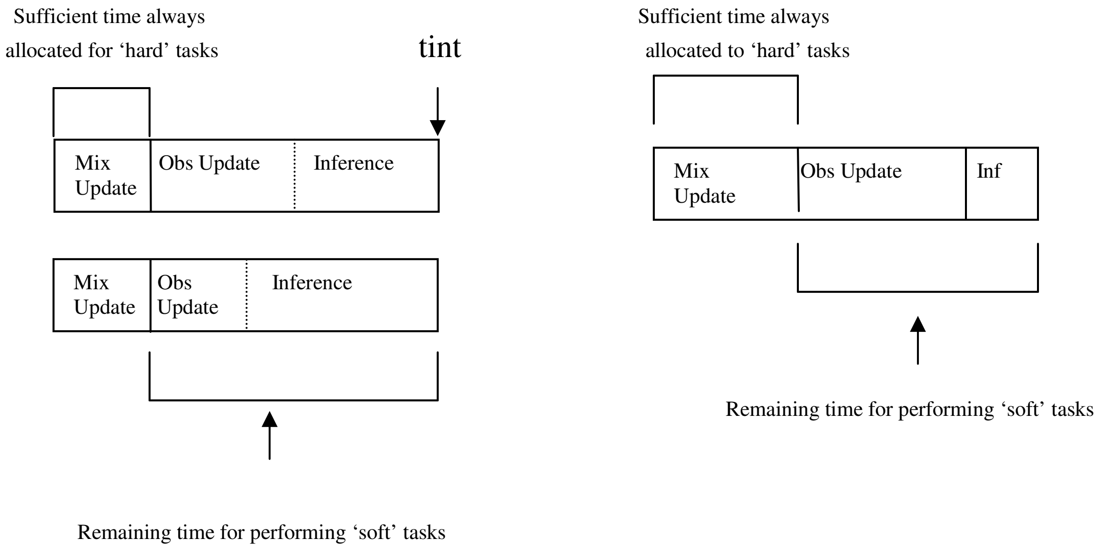

Sufficient time always allocated for ‘hard’ tasks
tint
Mix Update
Obs Update
Inference
Mix Update
Obs Update
Inference
Remaining time for performing ‘soft’ tasks
Sufficient time always allocated to ‘hard’ tasks
Mix Update
Obs Update
Inf
Remaining time for performing ‘soft’ tasks

Figure 24: Composition of soft tasks in terms of time allocated to each component is dynamic

Given a specific program structure, it is essential to always allocate sufficient time to execute to completion all identified “hard” components. Having done that, we need a mechanism to determine the optimal allocation of the remaining time among the “soft tasks”. Figure 24 illustrates this scenario in the context of our system prototype. The figure shows that there is always sufficient amount of time allocated to execute the Mixing Update task to completion. However based on the actual time taken by the Mixing Update task each interval, the remaining time before the deadline varies. The problem is to determine the optimal composition of “soft tasks” so that the remaining time before a deadline can be utilized in the best possible way to maximize the system solution quality at the deadline. Since we can independently choose the amount of time to be given to each of the soft tasks (Observation Update and Inference), the performance profile that we need to make the decision as to how best to utilize the remaining time is a multidimensional performance profile that indicates how the system solution quality changes for different allocations of time to the two “soft” sub-kernels.

We determine these multi-dimensional performance profiles for the two “soft kernels” in our reference implementation, based on various interleavings (time allocation/resources) of the ‘soft’ tasks. These composite performance profiles can subsequently be used to derive models that provide a means to determine the optimal composition of “soft” tasks given a specific resource/time constraint. We use this methodology of determining composite performance profiles for the “soft” kernels pertaining to our reference application.

5.4.1 Obtaining Composite performance profiles for soft kernels:

In order to determine performance profiles for various interleavings of time allocation (resources) provided to the “Observation Update” and “Inference” tasks, we decompose the tasks into the following granularities:

- Granularity of decomposition of the inference task: The grain size for an inference corresponds to an iteration of simulated annealing. We vary the number of iterations from 1 to MAX (MAX= maximum number of iterations of simulated annealing in the original code)

- Granularity of decomposition of the update task: Considering one sensor update as the grain size for the interleaving experiments becomes intractable due to the huge number of sensor reports coming in each time step. Hence the grain size of sensor updates is at a coarser level. Here is how the interleaving points for time (resource allocation) are obtained:

    1. Apply 5-100% mixing sensor updates each time step in steps of 5%
    2. Then apply 100% mixing sensor updates + (5-100%) sensor updates pertaining to similar entities in steps of 5%
    3. Then apply 100% mixing sensor updates + 100% sensor updates pertaining to similar entities + (5-100%) first time sensor updates in steps of 5%
    4. Then apply all updates.

This corresponds to 61x25=1525 sample points /inter leavings of sensor updates and inference for which we run the IDA algorithm (the Mix Update task is always executed to completion). Each of the experiments is run for a simulation window of 500 time steps. We determine the average accuracy and execution time values for each of the sample points to obtain the composite performance profile for the Observation Update and Inference tasks. Figure 25 shows the performance profile so obtained for a dataset of 3000 entities simulated over 500 time steps.

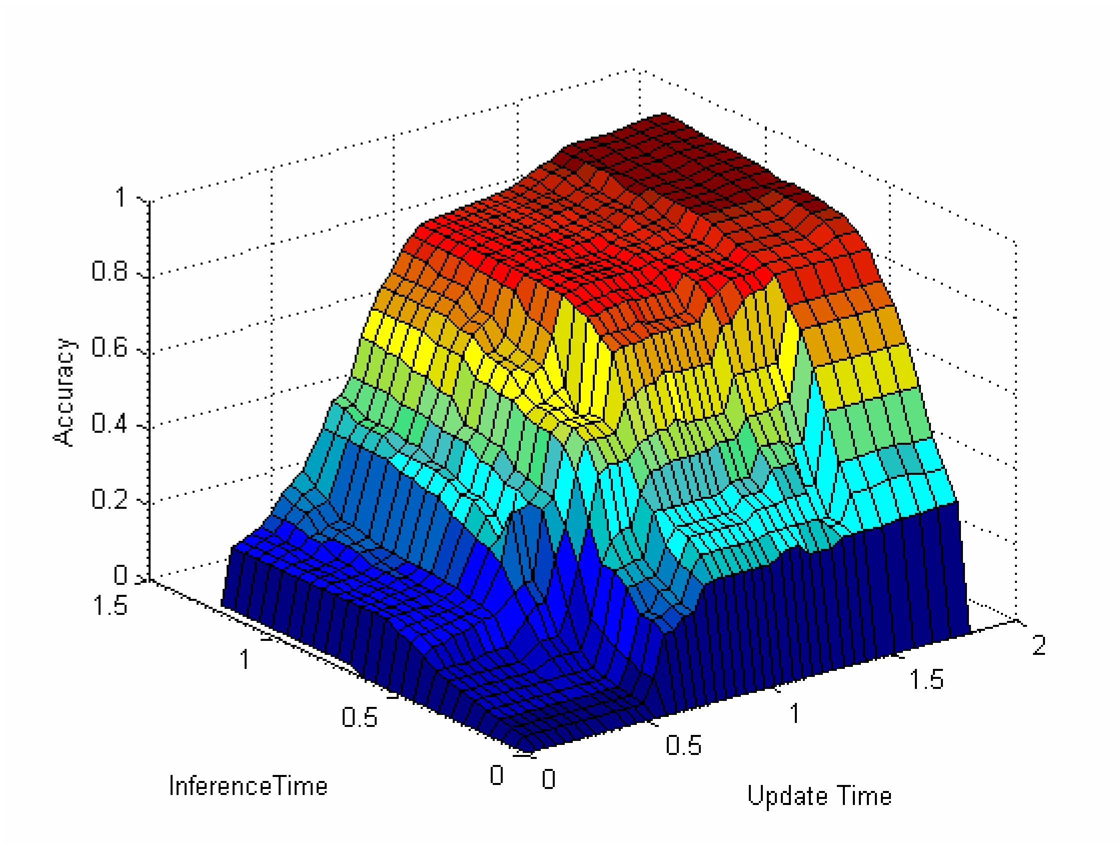


Figure 25: Composite performance profile for inter leavings of the update and inference task

5.4.2 Model for optimal composition of "soft" tasks:

Having derived the composite performance profile for the "Observation Update" and the "Inference" tasks, we derive a model that provides the optimal time allocation to these individual tasks given a specific amount of time in which to perform both so as to maximize the accuracy.

To derive this model we use a statistical analysis tool called "JMP", which is a widely used statistical analysis tool both in academia and industry[61]. We use the curve fitting facility provided by JMP to build an appropriate model for our performance profile.

The model thus derived for the performance profile in Figure 25 is a fourth degree polynomial given by:

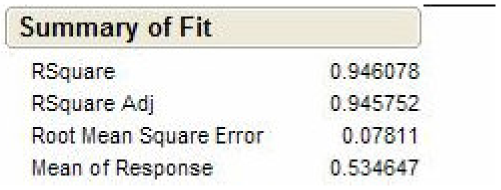
$$z = -0.36515 + 1.0464*x + 1.3283*y - 0.3151*x*x - 0.6955*y*y + 0.99209625*(y-0.60192)*(x-0.84122) - 0.3448163*(x*x-0.94611)*(y*y-0.49439) \qquad (1)$$

Where z = Tracking Accuracy

x = Observation Update Time

y = Inference Time

Having obtained the model, we evaluate the goodness of the fit provided by the model. The tool output, in addition to providing us the model parameters also provides a summary of the fit. The summary of the fit pertaining to the above model is given below:



Summary of Fit	
RSquare	0.946078
RSquare Adj	0.945752
Root Mean Square Error	0.07811
Mean of Response	0.534647

Table 1: Summary of the fit for the model derived from the performance profile

From the above summary, the goodness of the fit is evaluated as per the evaluation methodology outlined in [61]. The RSquare value in the fit summary in statistical terms is called the correlation coefficient and it represents the goodness of the fit. The table below obtained from [61] provides an overview for interpreting the meaning of the RSquare value

Correlation Coefficient	Descriptor
0.0-0.1	trivial, very small, insubstantial, tiny, practically zero
0.1-0.3	small, low, minor
0.3-0.5	moderate, medium
0.5-0.7	large, high, major
0.7-0.9	very large, very high, huge
0.9-1	nearly, practically, or almost: perfect, distinct, infinite

Table 2: Interpretation of the correlation coefficient,Rsquare in the context ,of goodness of the fit

The following optimization problem is then solved using the standard solver in Excel to obtain the optimal allocation of time to the Inference and Update Tasks given a specific remaining amount of time.

Maximize z :

subject to x+y <= R; x>0; y>0 (R= total time available for allocation between the Observation Update and Inference tasks. This essentially represents the remaining time to a deadline after the completion of the Mix Update task each time step interval. Henceforth in all the graphs this time is referred to as "Remaining time")

For the model represented by (1), Figure 26 is the three dimensional plot that shows the optimal time allocations to Inference and Observation Update tasks given specific time constraints. Interpretation of this plot in 2-D would be two graphs that represent the fraction of time allocated to the Observation Update and Inference tasks respectively, for various total time allocations/remaining time. Figures 27 and 28 illustrate these two dimensional graphs that represent how the fraction of time allocated to the Observation Update and Inference tasks vary as a function of the remaining time. These graphs are based on data points obtained from the model derived from the composite performance profile shown in Figure 25.

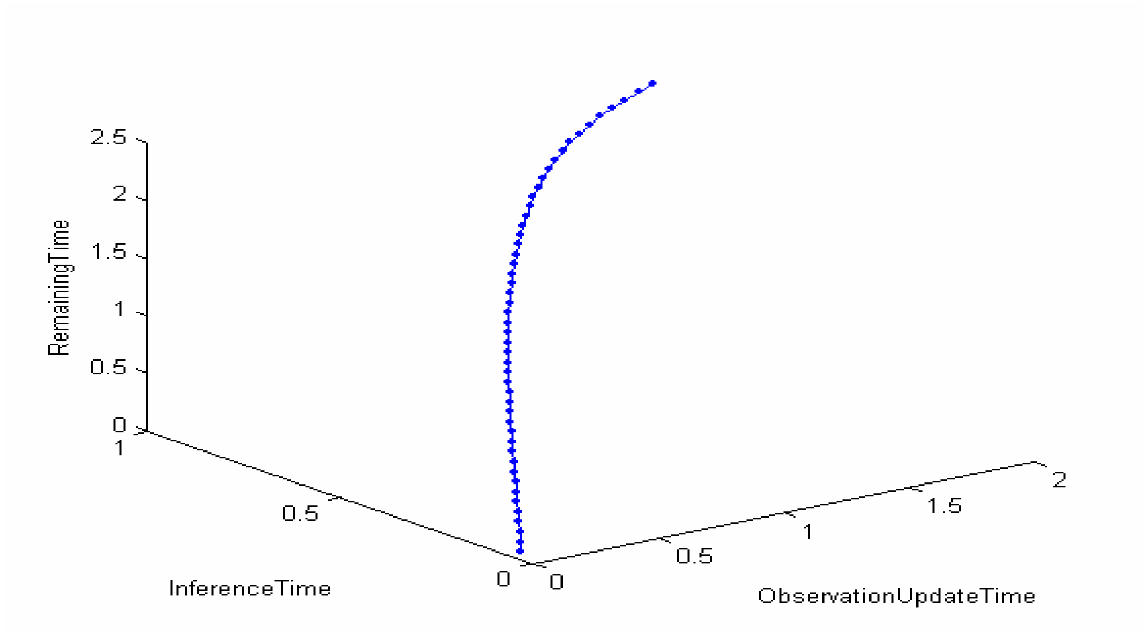


Figure 26: The projection of this plot on the x-y plane provides the optimal composition of Observation Update and Inference tasks given a specific total amount of time/resources to complete both.

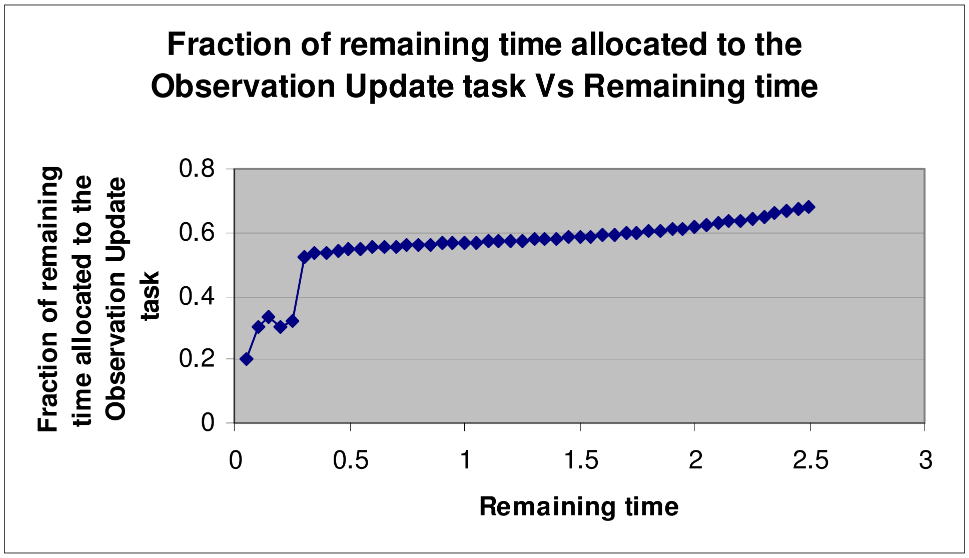


Figure 27. This figure shows the fraction of remaining time allocated to the Observation Update sub-kernel , given a specific amount of remaining time. This data has been derived from the model obtained from the performance profile shown in Figure 25.

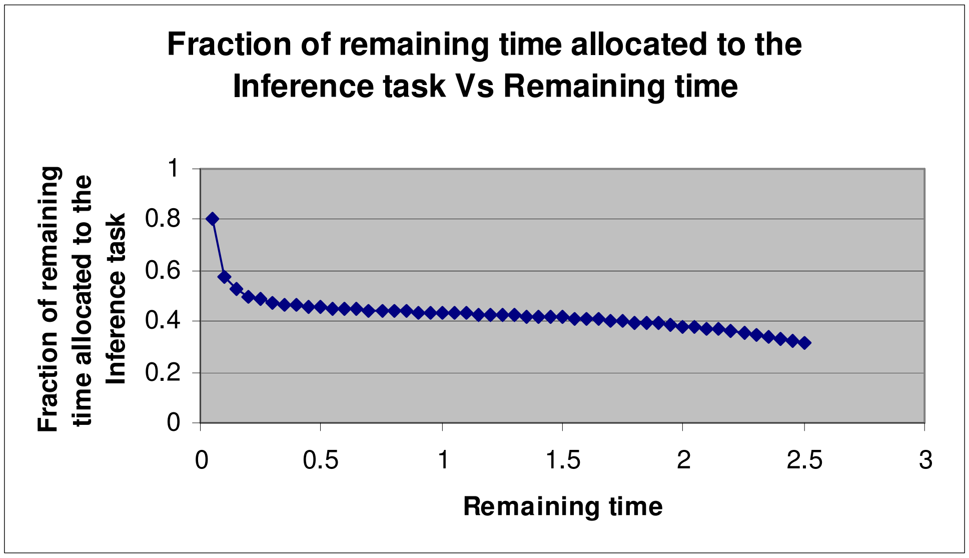


Figure 28. This figure shows the fraction of remaining time allocated to the Inference Update sub-kernel , given a specific amount of remaining time. This data has been derived from the model obtained from the performance profile shown in Figure 25.

Chapter 6

6. Scheduling Policies

As evident from our discussions and experiments described in the previous chapters, soft computations provide the flexibility to trade off accuracy for computation time in order to facilitate meeting real time deadlines with sufficient solution quality instead of designing a system to support worst case program execution. To exploit such opportunities in a real time system, some form of run time scheduling support is important. The run time component/scheduler needs to address a decision problem involving the choice of a collection of tasks to execute so the `best possible' outcome ensues for the system. At various decision points (at run-time), there are a set of tasks that are available for execution. In the absence of enough resources to execute all tasks to completion, a decision must be made as to the optimal composition of the set of tasks ready for execution at that point of time. This decision may involve picking out the `extra' tasks to support when resources are spare, or which services to sacrifice and to what extent when resources are scarce.

The models for optimal composition of soft tasks that we discussed in the previous section could be one form of input to the run time scheduler that facilitates decision making. However there are design issues in this context that need to addressed. More specifically,

- the decision points when the run time component needs to be invoked should be identified so as to ensure minimal scheduling overhead and optimal system utility.

- it is also essential to identify enough variance in the system to justify runtime support as opposed to determining a fixed composition of tasks pre run time that provides a sufficiently good system solution .

In this chapter, we address these issues and describe a dynamic scheme to schedule soft computations in the context of our real time system prototype. In our scheme, we use models derived as per the methodology outlined in the previous chapter to drive decisions about the optimal composition of tasks at specific decision points at run time. In Chapter 7, we shall present experimental results comparing our scheme to a "static soft" scheme and the conventional Earliest Deadline First scheme. Both these schemes are also described in detail in this Chapter.

Just to recapitulate, Figure 29 shows the basic problem formulation in case of our reference real time scenario. Every 'tint' units of time, a new set of sensor reports needs to be processed . The objective is to be able to maximize the accuracy of the system at the user refresh interval. Figure 30 shows the basic program structure of our application that processes the sensor reports.

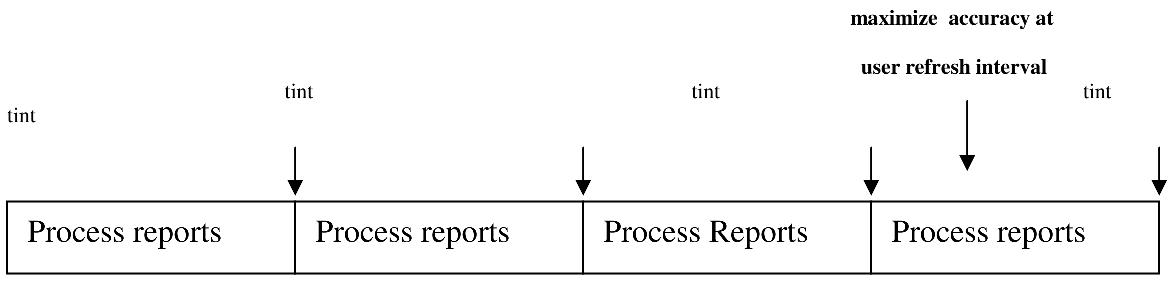


Figure 29: General problem formulation

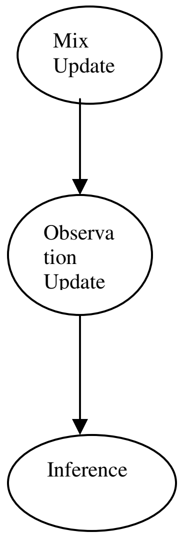


Figure 30: Program structure in our system prototype. The “Mix Update” task has been identified as “hard” while the “Observation Update” and “Inference” tasks have been identified as soft. Each of these tasks need to be performed every time step.

6.1 Static soft policy

We call the first scheduling policy that we apply to our system the “static soft policy.” This scheme assumes that a detailed timing analysis of all “non-soft” tasks has been done so that we can ensure that all “non-soft” tasks (in our case the Mixing Update/ MIX task) always run to completion. This scheme uses the models derived from performance profiles for soft tasks off-line to determine the optimal composition of soft tasks assuming the remaining time available for execution after scheduling the hard components and before the deadline will always be fixed for a given time step interval size, ’tint’ . In other words, this scheme works under the assumption that the variance in the “non-soft”/”hard” components of the system is very small with reference to it’s estimated maximum estimated execution time (we use the term maximum execution time here in place of worst case execution time because our timing analysis is based on statistics collected from application runs and no methodological worst case analysis has been done taking into account other important factors like the environment model). The objective of applying this policy is to understand if the dynamics in our reference implementation, more specifically the variance in the execution time of the "MIX" task, justifies run time monitoring or is it sufficient to exploit the trade-off opportunities provided by the "soft tasks" off-line. Figure 31 illustrates the inputs and outputs of the off-line analysis component. The scheduler uses the composition of "hard" and "soft" tasks obtained from this analysis to schedule the tasks every time step. This is illustrated in the Figure 32.

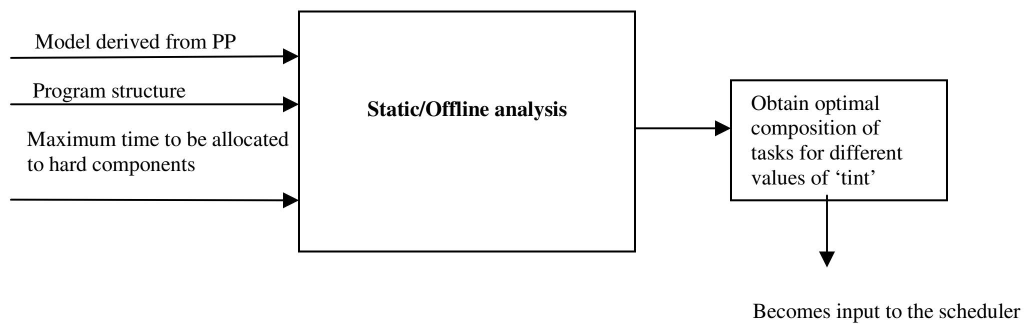
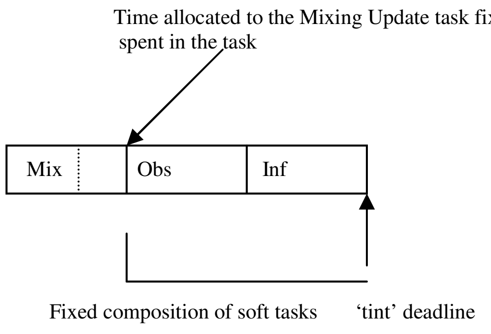

Model derived from PP
Program structure
Maximum time to be allocated to hard components
Static/Offline analysis
Obtain optimal composition of tasks for different values of 'tint'
Becomes input to the scheduler

Figure 31 Inputs and Outputs of the static analyzer


Time allocated to the Mixing Update task fixed irrespective of actual time spent in the task
Mix
Obs
Inf
Fixed composition of soft tasks
'tint' deadline

In all timeline diagrams:
Mix: Pertains to Mixing Update/MIX task

Obs: Pertains to OBSERVATION UPDATE task

Inf: Pertains to INFERENCE task

Figure 32 Same schedule shown in the figure is repeated every interval once optimal composition of soft tasks is determined offline based on maximum expected time spent in the MIX task.

6.2 Dynamic soft policy

This policy aims to exploit the dynamics of the system better by monitoring the execution time of the "non-soft" MIX task in the system prototype while exploiting the soft computing properties of the "OBSERVATION UPDATE" and "INFERENCE" tasks. The run time component is initialized with two inputs: the model obtained from the performance profile of soft tasks pertaining to the dataset (as described in Chapter 5), and the cost of executing each soft task at its finest granularity (in our case that pertains to the cost of a sensor update and the cost of an inference iteration). The model is fed into the run time component in the form of a table. Each entry of the table has two entries pertaining to the optimal time allocation to the update and inference tasks corresponding to a given total time allocation. Each entry in the table corresponds to a specific total time allocation.

One design issue is to make a decision about how big the table should be in terms of number of entries. The other design issue is to determine the granularity of the discrete intervals of total time for which we have entries in the table. We adopted the following approach to handle these issues. The granularity of time allocation is obtained (discretization of time is done) by starting with a total time allocation corresponding to the larger of the two values of the cost of a sensor update and the cost of an inference iteration and then increasing it by the same value for corresponding entries of the table. For example: if the cost of an inference iteration is 0.05 and that of a sensor update is 0.009, the starting total time allocation for the first table entry is 0.05 and thereafter corresponding entries are incremented by the same value. The size of the table is determined by using the performance profile curve fit to determine the total time allocation required to achieve maximum accuracy. This total time allocation is then divided by the smallest discrete time allocation unit to determine the total number of table entries. This methodology works well because the complexity of indexing the table is an O(1) operation and this also makes sure that the size of the table does not increase beyond what's needed to keep the run time overhead minimal. We also tried using as the time granularity the smaller of the two values of the cost of a sensor update and the cost of inference iteration. However, the corresponding benefits were not significant; hence we stuck to our original, more efficient design parameter.

Another design decision that needs to be made is regarding the decision point for the run time control to determine the optimal allocation of remaining time before a deadline to different tasks. This decision point in the context of our reference scenario is when a 'MIX' task is completed every time step. At that point of time, the run time component determines the time remaining before the deadline, accesses the table mentioned above to determine allocation of time to the 'OBSERVATION UPDATE' and 'INFERENCE' tasks, and by using the cost factors for each task, determines the number of sensor updates and inference iterations to be performed. This design decision is made based on the program structure of our application.

The other factor that needs to be taken care of is determination of a "safety margin" while making a decision regarding the composition of a set of tasks in the time remaining before a deadline. In our case, for all the datasets the safety margin is equal to the sum of the average execution time of inference iteration and the average execution time of a sensor update pertaining to that dataset. This has been observed to suffice in all our experiments to avoid deadline misses due to uncertainty in measurements. We also measured the average time spent in the run time component across various time steps, but the time so measured was miniscule compared to the sum of the average execution times of sensor update and inference iteration. Hence the run time overhead factor was not taken into account while determining the safety margin. However there has not been any detailed statistical analysis done to determine this margin. In the literature survey we carried out, we did not come across any methodological way of determining safety margins for systems (most of the methods we came across were ad-hoc and very domain specific [57]). Figure 33 illustrates the operations performed by the run time control each time step as soon as the 'MIX' task completes.

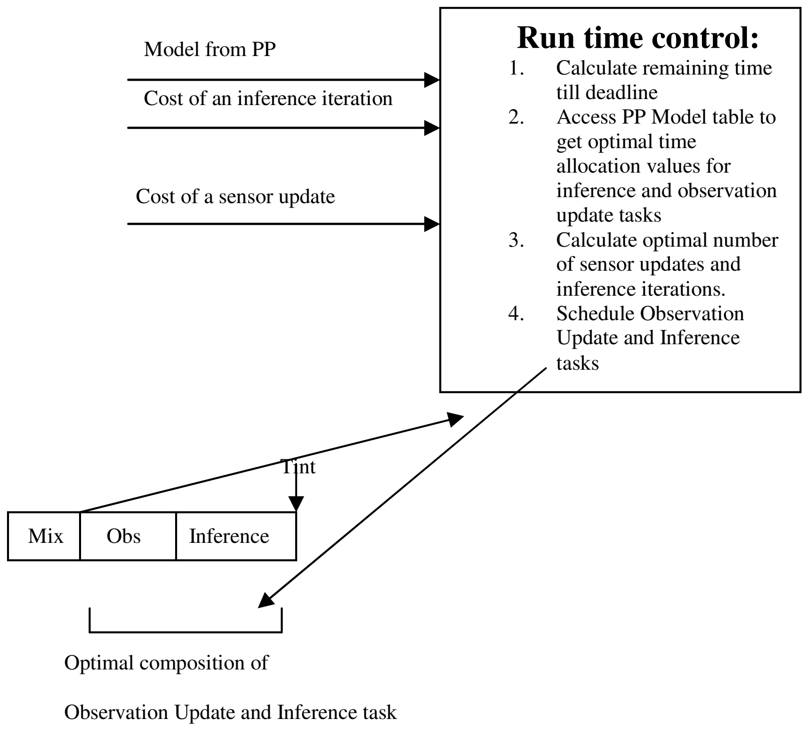

Model from PP
Cost of an inference iteration
Cost of a sensor update
Run time control:
1. Calculate remaining time till deadline
2. Access PP Model table to get optimal time allocation values for inference and observation update tasks
3. Calculate optimal number of sensor updates and inference iterations.
4. Schedule Observation Update and Inference tasks
Tint
Mix
Obs
Inference
Optimal composition of
Observation Update and Inference task

Figure 33 Basic operations performed by the run time control after the completion of the MIX task each time step interval

6.3 Conventional Dynamic Earliest Deadline First (EDF) policy

The Earliest Deadline First (EDF) scheduling policy dynamically assigns priorities to tasks as they arrive based on their absolute deadline values. We perform an execution trace based analysis of a non preemptive version of the Earliest Deadline First scheme as applied to the tasks in our real time scenario.

The EDF formulation for our scenario is done as follows: All three tasks arrive every time step ; the deadline of MIX and OBSERVATION UPDATE tasks is equal to the time step interval but the deadline of the INFERENCE task is equal to the average case execution time of (MIX+UPDATE+INFERENCE). By having a separate deadline for INFERENCE, we can ensure that an UPDATE task that completes beyond the time step interval deadline(becomes tardy) does not prevent an INFERENCE from being performed as long as the UPDATE completes before the deadline for the INFERENCE task for the corresponding time step. This formulation ensures better system solution quality rather than formulating the problem with all three tasks having the same deadline. All tasks are either run to completion (we allow tasks to be tardy) or if their deadlines have been crossed before they are provided CPU execution time, the tasks are dropped.

Let ‘tint’ be the time step interval size/deadline for OBSERVATION UPDATE and MIX task and let ‘infdeadline’ be the periodic deadline for INFERENCE. Table 3 shows the task characteristics assumed in our EDF formulation.

Tasks	Arrival	Period	Deadline
MIX	T1,t1+tint,t1+2*tint,t1+3*tint,….	Tint	t1+tint,t1+2*tint,t1+3*tint,….
UPDATE	T1,t1+tint,t1+2*tint,t1+3*tint,….	Tint	t1+tint,t1+2*tint,t1+3*tint,….
INFERENCE	T1,t1+tint,t1+2*tint,t1+3*tint,….	Tint	t1+infdeadline,t1+2*infdeadline,t1+3*infde

Table 3: This table shows the tasks characteristics for the EDF formulation

For each dataset, we obtain a trace of the execution times for the different tasks in the different time steps. Based on this execution trace, for different values of 'tint', we perform an analysis as to what tasks cannot be executed owing to the tardy completion of earlier tasks. Even if a task misses a deadline, we assume that the task runs to completion .The set of tasks obtained after excluding the tasks that get dropped are then executed in time step order to obtain corresponding accuracy values. Figure 34 illustrates an EDF schedule obtained using this methodology when the tasks have deadlines and execution times as shown at the top of Figure 34.

	Exec Times	Deadlines	
MIX1	0.3	1.5	Meets deadline
UPDATE1	1.5	1.5	Tardy
INFERENCE1	1.2	2.4	Meets deadline
MIX2	0.4	3	Tardy
UPDATE2	1.3	3	Dropped
INFERENCE2	1.2	4.8	Dropped

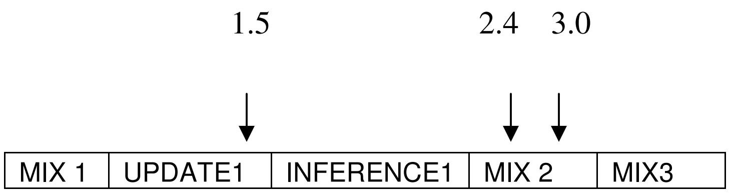


Figure 34. An EDF schedule for representative tasks in two consecutive time steps

Chapter 7

7. Experimental Results

In this chapter, we discuss the experimental results obtained by applying the scheduling policies described in Chapter 6 to our reference scenario. We first discuss the metrics we use for our evaluations. Thereafter, we explain our experimental set up and finally we present the experimental results obtained on various data sets. In addition to this, we present the results of the sensitivity study we carried out in order to determine the sensitivity of the performance profiles we use to two different input data parameters.

7.1 Evaluation metrics

In order to quantify the benefits of our scheduling methodology, we use the following three metrics (We shall discuss how they are measured in our experimental framework in section 7.2):

- Tracking accuracy measured at each user defined refresh interval
- CPU utilization
- Fraction of missed deadlines

Our aim is to determine how the tracking accuracy of the system, the CPU utilization and the fraction of missed deadlines vary as we increase the time/resources provided to the application to complete a given amount of work using the scheduling policies discussed in Chapter 6. Note that when deadlines are missed, the most recent tracking solution and data associations are used. This adversely affects the tracking accuracy.

So tracking accuracy is related to missed deadlines in addition to being related to the number and type of sensor reports that are processed each time step.

In the context of our reference scenario and system prototype, we explore the impact of different scheduling policies on the three metrics defined above as we increase the size of the time step interval without affecting the number and type of reports generated in each time interval. For each data set obtained from the NG test bed, we perform a set of experiments by progressively relaxing the 'tint' deadline constraint across experiments, keeping the amount of work needed to be done each 'tint' interval the same as the original dataset. Varying the time interval size in this manner can be considered equivalent to varying the speed of the CPU given the fact that since our setup is running on fixed piece of hardware, there is no other simple means of emulating variation in CPU speed. By measuring the tracking accuracy and missed deadlines as a function of time interval size, we evaluate how aggressive a real time constraint our reference CPU can meet across different schedulers. A scheduler is better than another if it can meet a tighter real time constraint at an equivalent level of functional correctness.

When we adopt this methodology to evaluate our scheduling policies, keeping the user defined refresh interval fixed at an absolute value of wall clock time creates a problem. If we keep 'trefresh' fixed at an absolute value, as we increase 'tint' the total amount of work needed to be done by the processor (in terms of sensor report processing) before hitting the 'trefresh' deadline decreases since the ratio of 'trefresh' to 'tint' decreases. Hence in order to make a fair comparison, instead of making 'trefresh' a deadline in terms of absolute time, we consider it to be a deadline in terms of an interval number. Now the system objective changes to maximizing the tracking accuracy at the end of every‘t’ intervals instead of ‘trefresh’ units of time. This translates to maximizing the tracking accuracy every‘t’ intervals of time by meeting each of the ‘tint’ deadlines corresponding to the sensor report arrival rate in the best possible way. This is the framework within which we evaluate the three scheduling policies discussed in Chapter 6.

7.2 Experimental set up

All our experiments have been performed on a 3GHz dual processor Xeon machine on a Linux platform. All execution time numbers are single thread performance numbers on this machine, so we make use of only one CPU. Table 4 provides a full description of the hardware set up we use.

Processor	3GHz Xeon
L1 cache size	16kB
L2 cache size	2MB
Main memory	3GB
Front Side Bus	800 MHz
Operating System	Red Hat Linux Ent V4.0 Kernel V2.6

Table 4: Hardware set up

The customized scheduler /run time control component in our framework has been developed as a component of the application and hence is a part of the same process/address space as the application tasks. Details about the run time control component/scheduler have been discussed in Chapter 6

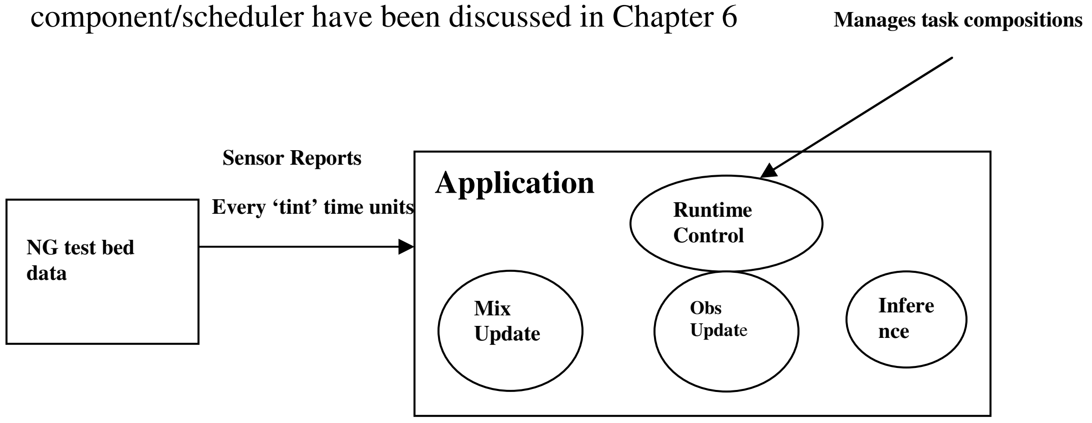


Figure 35: Basic framework of the experimental set up

7.2.1 Data sets used

The scheduling policies are evaluated on 6 different data sets generated from the Northrop Grumman test bed by performing simulations using different configuration files. Another two additional data sets are used for the sensitivity study. The 6 data sets used for the scheduling policy evaluations differ in terms of number of entities and have the default behaviors as defined in the NG test bed corresponding to the entity classes to which they belong. The additional two data sets that we use for the sensitivity study differ in terms of the behaviors associated with the entity types. All other parameters in the configuration files are kept the same while generating these data sets. Figure 37 shows the dataset generation framework. Table 5 shows the different parameters pertaining to the datasets.

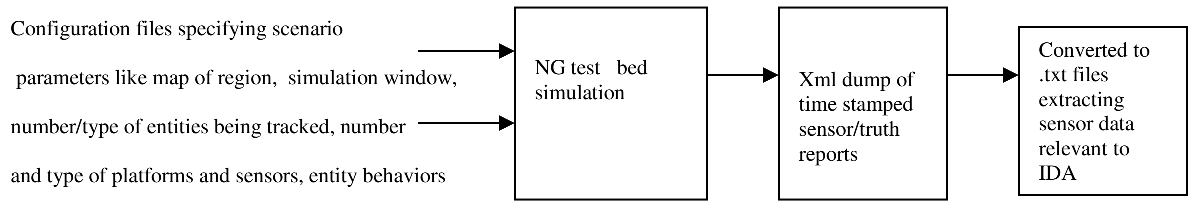

Configuration files specifying scenario parameters like map of region, simulation window, number/type of entities being tracked, number and type of platforms and sensors, entity behaviors
NG test bed simulation
Xml dump of time stamped sensor/truth reports
Converted to .txt files extracting sensor data relevant to IDA

Figure 37: Data set generation from the NG test bed

Data set parameters	**Value**
Number of types of entities	**50**
Number of attributes / features per entity type	**10**
Number of platforms	**55**
Number of sensors / platform	**3**
Number of entities	**1000, 1500, 2000, 3000, 4000, 5000**
Behavior sets	**Each entity type has an associated behavior specified by start time, end time, start location, end location and one of several routes from start location to end location which includes the number of route points where the entity stops en route to the destination. The commute time and the pause time are also specified. We use a set of behaviors that are default and another synthetically generated set of behaviors for our sensitivity study.**

Table 5 Parameters pertaining to the data sets

7.2.2 Simulation of real time sensor report delivery to the application

The real time sensor report delivery to the application is simulated using UNIX timers and signals. At initialization, the application reads the input .txt file containing the sensor reports and dumps them into a linear array in memory. Initially only the set of sensor reports pertaining to the first time step interval is accessible to the tasks. Once all other application initializations are completed, just before calling the scheduler to start the scenario simulation, the application sets a timer with an interval equal to the time step interval size 'tint' pertaining to the experiment. Thereafter every 'tint' units of time, the application is asynchronously interrupted by a SIGALRM signal. The signal handler in the application corresponding to this signal updates a global time stamp value each time the SIGALRM signal is caught. This update of the timestamp value signifies the arrival of a new set of sensor reports corresponding to a new time step that now becomes available to the tasks in the application. Since signals work like software interrupts, immediately after returning from the signal handler, the application/task resumes operation at the point where it was interrupted.

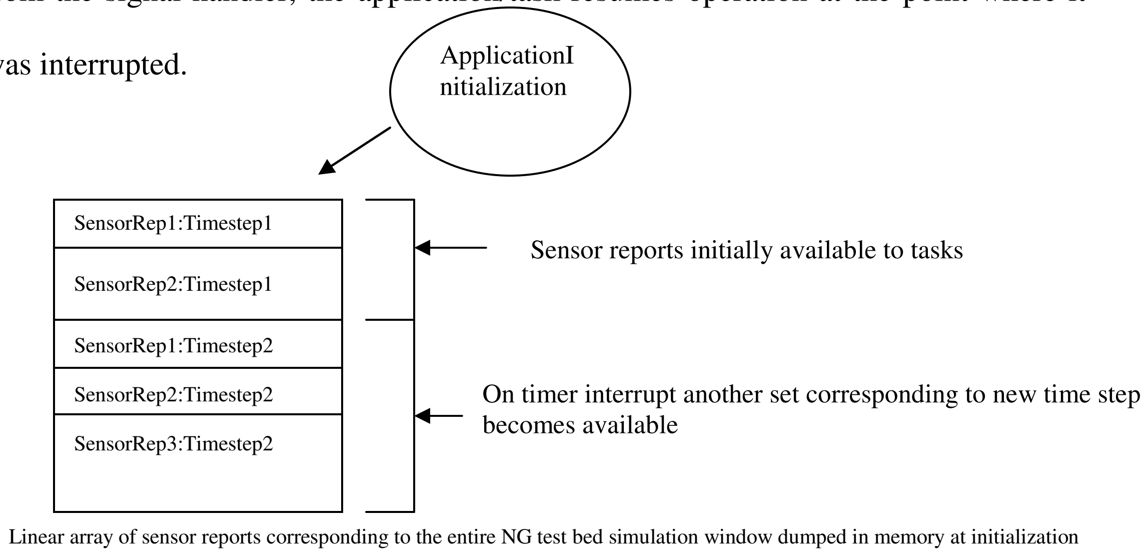

ApplicationI nitialization
SensorRep1:Timestep1
SensorRep2:Timestep1
SensorRep1:Timestep2
SensorRep2:Timestep2
SensorRep3:Timestep2
Sensor reports initially available to tasks
On timer interrupt another set corresponding to new time step becomes available
Linear array of sensor reports corresponding to the entire NG test bed simulation window dumped in memory at initialization

Figure 38: Simulation of real time sensor report delivery to application

In this framework of simulation of real time sensor data delivery to the application, we consider the availability of an infinite buffer for the sensor reports. However as will become apparent in later discussions, it will suffice to buffer two sets of sensor reports at a time, one pertaining to the current time step interval and the other pertaining to the previous time step interval. This is to facilitate "tardy" tasks corresponding to the previous time step to complete successfully when using our dynamic scheduling scheme. In our current set of results, using our dynamic scheduling scheme, there are no deadline misses because we use a "safety margin" while scheduling tasks to minimize the chances of tasks becoming tardy. Even if tasks do become tardy, the likelihood of them crossing more than two time step boundaries is nominal using our scheme.

7.2.3. Counting missed deadlines and measuring CPU idle time

If the SIGALRM signal interrupts the application when one of the three application tasks is running, on return from the signal handler, the current task and subsequent tasks pertaining to that time step are run to completion. Once the set of tasks is completed, the time stamp associated with the completed task set is compared to the most recent timestamp updated by the signal handler. If the timestamps differ, a deadline miss is recorded. If the timestamps are the same, the time remaining for the timer to expire is obtained from the system and that becomes the CPU idle time for that time step. Figure 39 illustrates this methodology.

CPU utilization is calculated at the end of the simulation as:

$$\frac{\text{'tint'*num\_sim\_timesteps} - \Sigma\text{CPU Idle Time}}{\text{'tint'*num\_sim\_timesteps}}$$

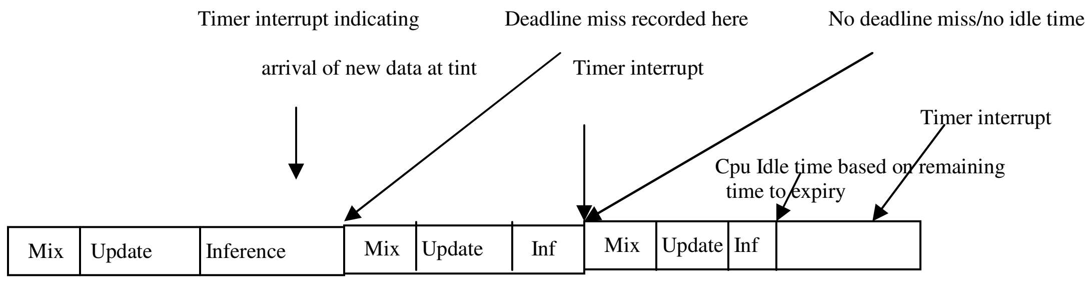

Timer interrupt indicating
arrival of new data at tint
Deadline miss recorded here
Timer interrupt
No deadline miss/no idle time
Timer interrupt
Cpu Idle time based on remaining time to expiry
Mix
Update
Inference
Mix
Update
Inf
Mix
Update
Inf

Figure 39. Measuring CPU idle time and deadline misses

7.2.4. Measuring accuracy

The updated data associations obtained at the end of each inference task are dumped into a file at the end of the simulation. Off-line, we compare the data associations corresponding to the user defined refresh intervals to the ground truth corresponding to that interval. A score is obtained for the number of correct associations at each such interval. The scores are summed across all intervals pertaining to user refresh intervals and the average accuracy is so obtained.

7.3 Experimental results

7.3.1 Data set naming convention

We use the data sets mentioned in the Table 6 for the evaluation of our scheduling policies. Table 7 shows the additional two data sets used in the sensitivity study which we shall discuss in detail in section 7.4

Data set name	**Number of entities**	**Behavior set**
1000_1	1000	Default
1500_1	1500	Default
2000_1	2000	Default
3000_1	3000	Default
4000_1	4000	Default
5000_1	5000	Default

Table 6 Datasets used for the evaluation of our scheduling policies

Data set name	**Number of entities**	**Behavior set**
3000_2	3000	Changes made in default behaviors in terms of start time, end time, start location and end location
5000_2	5000	Changes made in default behaviors in terms of start time, end time, start location and end location

Table 7 Additional datasets used for sensitivity study

7.3.2 Variation in tracking accuracy with different scheduling policies

In this section we discuss our experimental results that show the impact of the different scheduling policies on the system solution quality /tracking accuracy

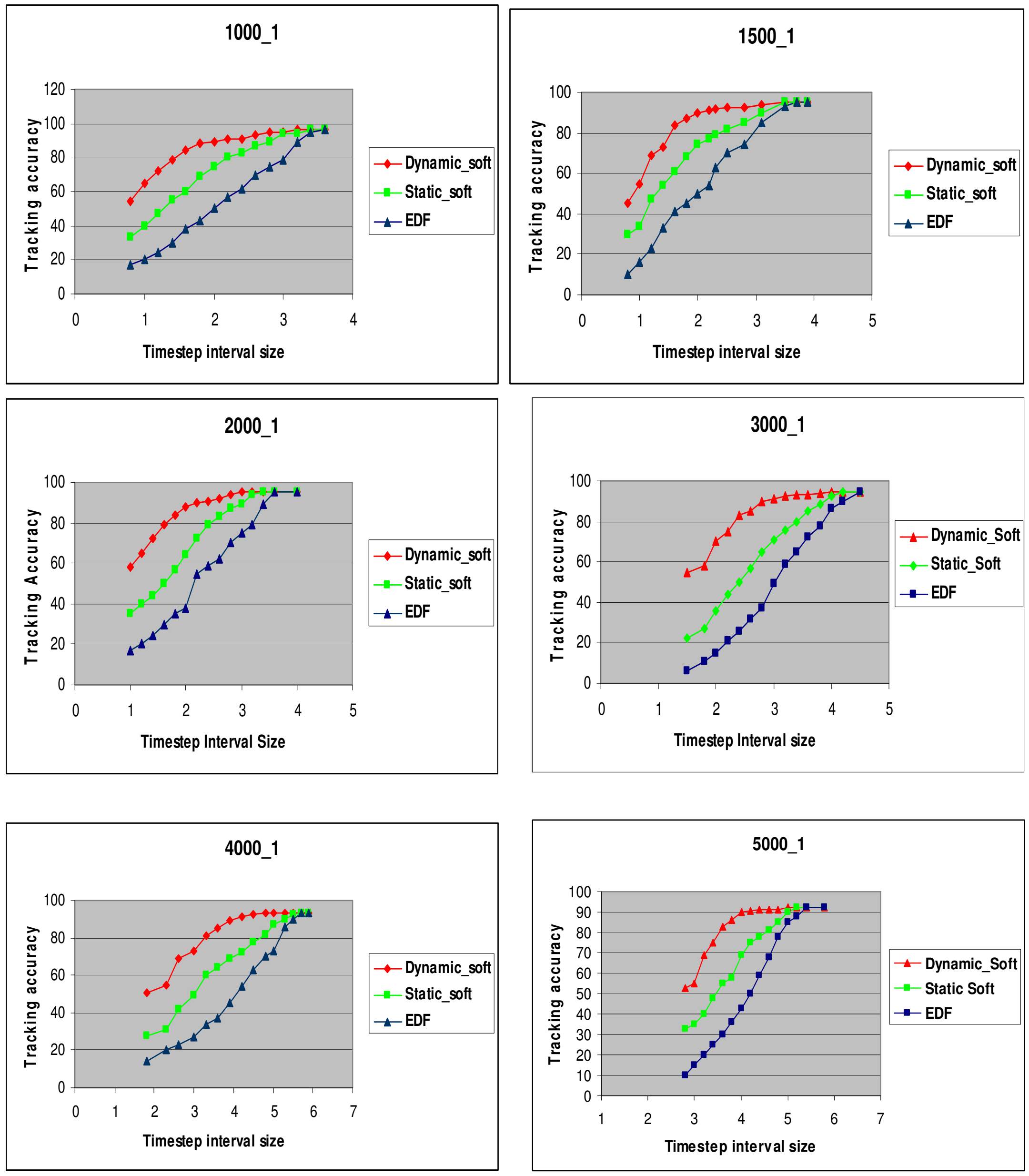


Figure 40 Graphs illustrating the impact of different scheduling policies on the tracking accuracy as time for completion of processing of each set of sensor reports i.e. the time step interval size ('tint') is increased for each dataset.

The graphs in Figure 40 illustrate the impact of the three scheduling policies discussed in Chapter 6 on the tracking accuracy of our application for different datasets as we vary the value of 'tint', the time step interval size, from a minimum value (pertaining to the maximum estimated time for mixing for that dataset) to a maximum value (pertaining to the maximum estimated time to perform all 3 tasks to completion every time step). The x-axis in all these graphs pertains to time step interval sizes and the y-axis pertains to tracking accuracy values.

We observe that the tracking accuracy obtained using our dynamic soft scheduling policy is on an average 26% better than the static soft policy and 39 % better than our implementation of the conventional Earliest Deadline First policy. The static soft policy on an average provides 13% better accuracy values than the conventional Earliest Deadline First policy. As evident from the figure, the tracking accuracy for all three scheduling techniques level off at higher values of time step interval sizes. This is owing to the fact that independent of what scheduling policy we use, the time pertaining to those time step intervals is high enough to ensure completion of all three application tasks every time step without missing the deadline. These points at which the three policies level off are included in the numbers we report above. If we do not include these points in our analysis, the difference in average tracking accuracies between the three policies is much higher.

The Earliest Deadline First policy does not perform well in the context of our reference application due to the fact that this policy tries to schedule each of the three tasks to completion every time step. This results in unpredictable missed deadlines, domino effects, tardy tasks and consequently a huge number of tasks need to be dropped at the admission control level due to the fact that their deadlines get crossed even before they are allocated CPU execution time. This dropping of complete tasks at the admission controller results in older and hence less accurate data associations (lower tracking accuracy) at the user defined refresh intervals.

The static soft policy performs better than the Earliest Deadline First because of the fact that instead of trying to execute all tasks to completion, this policy makes sure that the MIX task always runs to completion every time step but instead of trying to execute all soft tasks to completion before the deadline, it intelligently uses a predefined optimal composition of the OBSERVATION UPDATE and INFERENCE tasks to meet each deadline in the best possible way. However because the optimal composition of tasks is determined pre run time and the same composition is used every time step, this policy is unable to exploit the variance in the execution of the MIX task to get better solution quality.

The dynamic soft policy not only exploits the soft computing properties of the OBSERVATION UPDATE and INFERENCE tasks, but also monitors the execution time of the MIX task at run time. By doing so, for each time step, this policy is able to determine the optimal composition of INFERENCE and OBSERVATION UPDATE tasks at run time to meet the time step interval deadline .This ensures higher tracking accuracies at the user defined refresh intervals. The design point pertaining to the highest time step interval size in all the graphs is the design point corresponding to a conventional real time system design approach. Though the accuracy is maximum for this design point, the corresponding CPU utilization is very low as will become evident from the results in the next section.

Most importantly, the results for all the data sets illustrate the fact that using the dynamic soft scheduler; a much tighter real time constraint can be met than possible with conventional real time scheduling schemes for equivalent levels of functional correctness. For instance, for a data set 1500_1, the dynamic scheduler is able to achieve the same level of functional correctness at a time step interval size of 2 units (seconds) as opposed to a time step interval size of 4 units (seconds) required when scheduling tasks with a conventional scheduler. We conclude that with a dynamic soft scheduler, the tightest real time constraint (same as time step interval size) that can be met on our reference system and scenario is on an average 1.65 times smaller than what can be achieved by a conventional real time scheduling scheme at comparable levels of functional correctness.

7.3.3 Variation of CPU utilization with different scheduling policies

In this section we discuss our experimental results that show the impact of the three scheduling policies on the CPU utilization of the system.

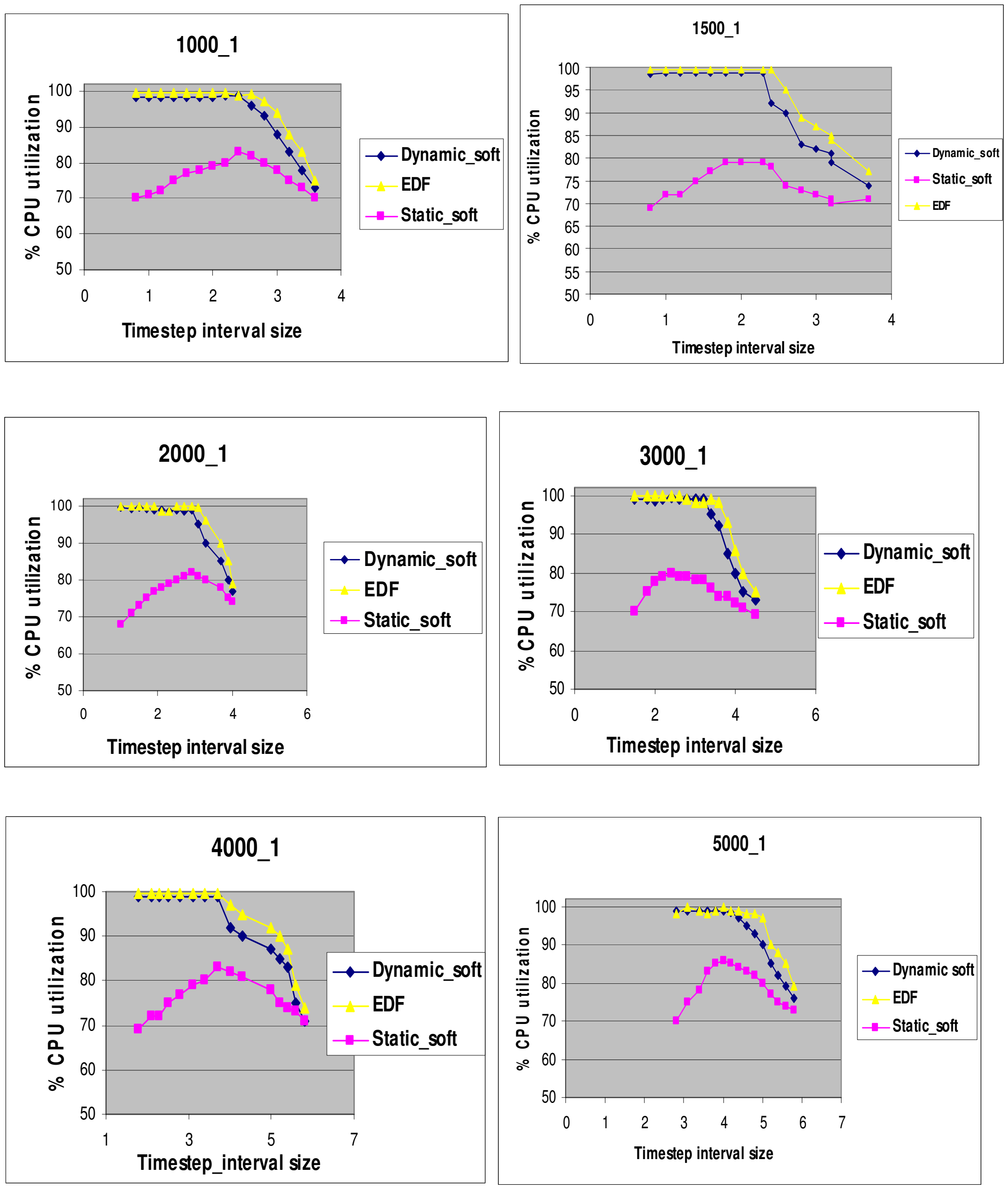


Figure 41: Graphs illustrating the effect of the 3 different scheduling policies on the CPU utilization of the system

Figure 41 shows the graphs that illustrate the impact of the three scheduling policies on the percentage CPU utilization of the system. In all the graphs, the x-axis pertains to the time step interval size and the y axis pertains to the corresponding CPU utilization.

For all the data sets, we observe that the CPU utilization for the static soft scheme is low at the beginning (smaller values of time step interval size), then starts increasing and finally decreases and tapers off again. Let us explain this with the example of the CPU utilization of the curve corresponding to dataset 2000_1. For this dataset, the CPU utilization initially is 68% corresponding to a time step interval size of 1 second. Since in the static soft scheme, the time allocated to the MIX task is always fixed (pertaining to the maximum estimated execution time for this task), the initial CPU idle time and low CPU utilization can be attributed to the variance in the actual execution time of the MIX task as compared to the time allocated to it.

Subsequently as the time step interval size is increased, the CPU idle time remains the same till a time step interval size of 2.8seconds. This means that there are always sufficient numbers of observation updates and inference iterations to be performed in each time step interval. This explains the increase in the CPU utilization values for the static policy from a time step interval size of 1second-2.8seconds. However as the time step interval size is further increased beyond 2.8 seconds, the variance in the number of sensor reports coming in causes the CPU idle time to increase further. This essentially means that when the time step interval time is higher than 2.8 seconds, there are some time steps where there are not enough sensor observation updates and inference iterations to keep the CPU busy for the allocated time. This is the cause of the subsequent decrease in CPU utilization of the static soft scheme.

For the same dataset, the dynamic soft scheme on the other hand starts with almost 100% CPU utilization, remains at this level till around the 2.7 second time step interval size and subsequently the utilization starts going down owing to the fact that for higher allocations of time, there are not always enough sensor observation updates and inference iterations to be performed in the allocated time. The dynamic soft policy essentially utilizes resources much more efficiently than the static policy and ensures that the system objective /solution value is maximized given a set of resources/time.

The other interesting fact in these graphs is that the CPU utilization pertaining to the EDF policy for all the datasets is also very high. However, the corresponding accuracy values for the EDF policy are not encouraging. This can be attributed to the fact that since EDF tries to schedule all tasks to completion, for a major part of time , the CPU is executing tasks that are tardy i.e. tasks that do not add much value to the final system solution by causing the unpredictable dropping of later tasks . Tardy tasks/dropped tasks and domino effects caused by EDF result in lower accuracies at the user defined refresh intervals. Thus with regards to handling the dynamics of the system, the important difference between the EDF policy and the dynamic soft policy is that the dynamic soft policy composes and schedules tasks "intelligently" by exploiting the soft computing properties of tasks to meet deadlines and hence maximizes the system utility while EDF does not. The design point pertaining to the highest time step interval size for each experiment is the design point corresponding to a conventional real time system design approach.

7.3.4 Variation in missed deadlines with different scheduling policies

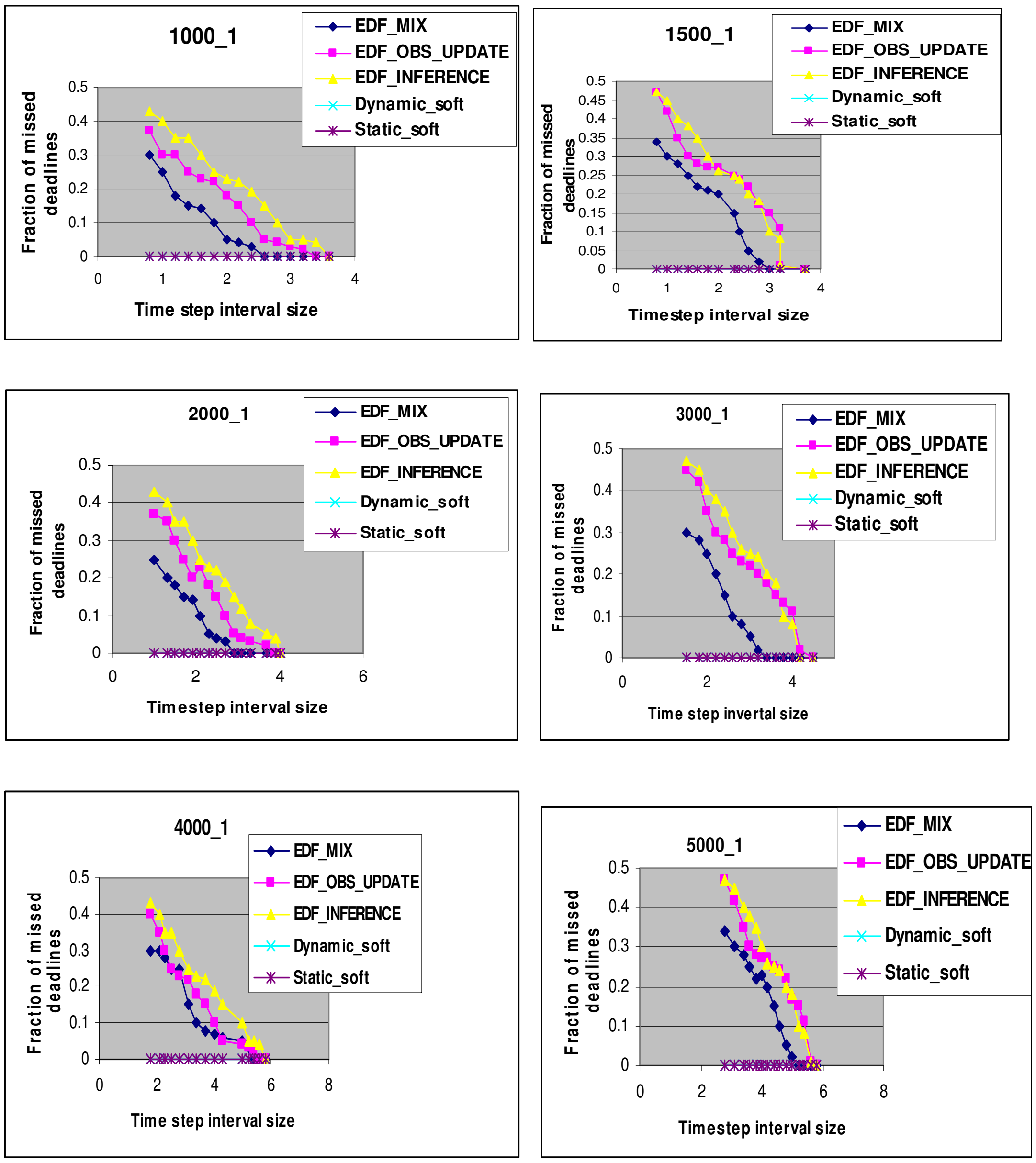


Figure 42: These graphs illustrate how the fraction of missed deadlines for each of the tasks varies across the scheduling policies

Figure 42 illustrates how the fraction of missed deadlines varies across the scheduling policies. In the figures, the x-axis pertains to the size of the time step interval and the y-axis pertains to the fraction of missed deadlines.

As evident from the graphs, for both “static soft” and “dynamic soft” policies, the number of missed deadlines is zero. This is because in both the static and dynamic policies, the decision about the composition of soft tasks is taken based on the remaining time to reach a deadline. Each time step interval is big enough to accommodate the worst case execution time for the “MIX” task, and the remaining time is allocated to a composition of the soft “OBSERVATION UPDATE” and “INFERENCE” tasks taking into account a safety margin so that the deadline is not missed.

In EDF however, the decision to schedule a task is solely made based on its absolute deadline and the remaining time to meet the deadline is not taken into account for scheduling tasks. Moreover each task runs to completion, even if it is tardy. As a result, the three tasks pertaining to the three sub-kernels “MIX”, “OBSERVATION_UPDATE” and “INFERENCE” miss deadlines unpredictably. Though the fraction of missed deadlines goes down for each of the tasks as we increase the time step interval, the missed deadlines cause later tasks to get dropped at the admission control level and hence tracking accuracy is adversely affected because the tardy and dropped tasks cause older data associations to be provided to the human operator at each refresh interval.

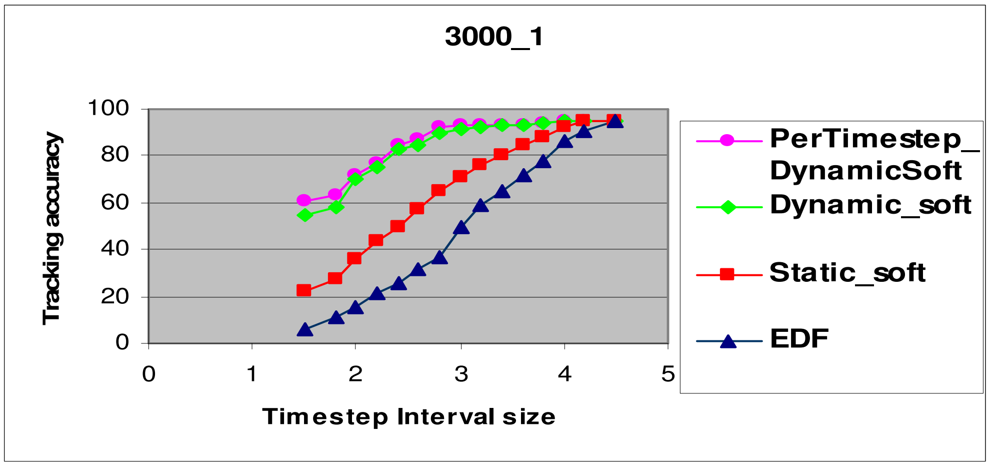
7.4 **Sensitivity study for performance profiles**

So far, in all our discussions, we have derived and used performance profiles pertaining to a specific data set by using average case numbers for the tracking accuracy and execution time across a large number (500) of simulation time steps. Subsequently, these performance profiles have been used for dynamic scheduling of tasks pertaining to the data set to meet various real time constraints. As evident from the results discussed in the previous section, the benefits are considerable. However it is important to explore the sensitivity of performance profiles to input data for a couple of reasons. Firstly we need to identify if the performance profiles we obtain based on average case accuracy and execution time numbers are representative for the entire dataset across all time steps or do we need per time step performance profiles to get more benefits from using the dynamic scheduler. Secondly, a sensitivity study also indicates if online learning of performance profiles is essential as opposed to using performance profiles that are determined off-line on a representative data set.

In this section, we discuss the experiments performed by us to understand the sensitivity of the performance profiles of the soft tasks in our reference implementation to various data set parameters. We study the sensitivity of performance profiles to input data in two phases: First we explore how the performance profiles for a single data set varies across the time step intervals. Secondly, we study how performance profiles vary across data sets as we change the number of entities and the behaviors associated with the entities. The effects of varying the number of entities and behaviors associated with them on the computations performed by IDA have been discussed in section 4.2.3

7.4.1 Use of dynamic performance profiles across time steps for a single data set

The aim of this experiment is to determine how per time step dynamic performance profiles vary from performance profiles obtained by considering average case execution time and accuracy values across time steps. For two of the data sets, we obtain per time step performance profiles for the first 100 time steps and then derive models from them as per the methodology outlined in Chapter 5. We observe that the variance in the dynamic per time step models and the model derived from the performance profile pertaining to average case numbers is not significant. We use these per step models as input to the run time control and compare the corresponding accuracy results with the results obtained using performance profiles pertaining to average case numbers. We find that for smaller values of time step interval size, there is around 5-6% gain in accuracy, but as we increase the time step interval size further, we observe diminishing returns in accuracy improvement. Figure 43 illustrates these results for two of the data sets: 3000_1 and 4000_1.



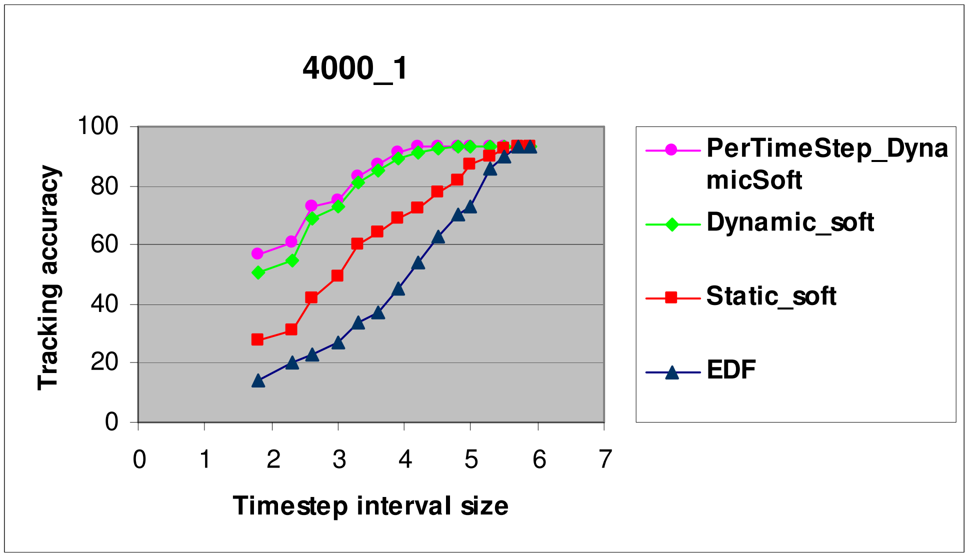


Figure 43. These figures show how tracking accuracy is affected when we use per time step performance profiles in place of performance profiles based on average case numbers.

We can tentatively conclude from the results that the performance profile derived by considering average accuracy and execution time numbers for various compositions of sensor updates and inference is a reasonable representative performance profile at least for this application.

7.4.2 Sensitivity to different number of entities

In this experiment, we apply performance profiles obtained for a data set with a specific number of entities on other datasets with smaller number of entities. The range of values in a model pertaining to a data set with smaller number of entities is a subset of the range in models pertaining to data sets with larger number of entities. Hence it is feasible to apply a performance profile pertaining to a dataset with larger number of entities to a dataset having smaller number of entities but not vice versa.

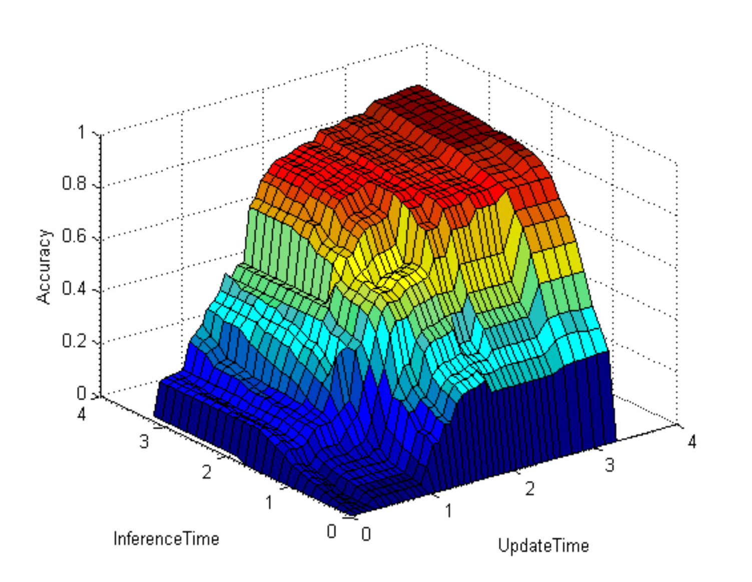


Figure 44: Performance profile for dataset 5000_1

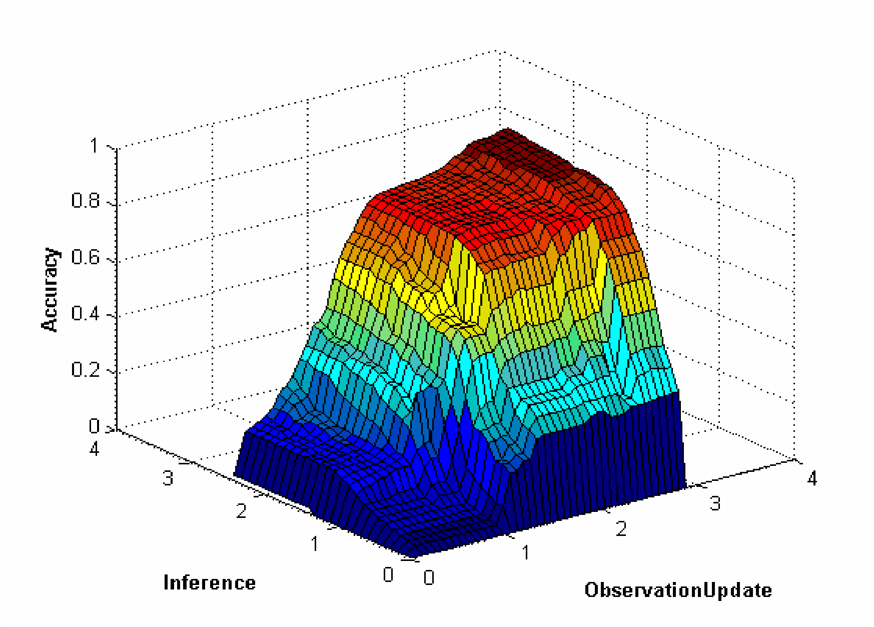


Figure 45: Performance profile for dataset 4000_1

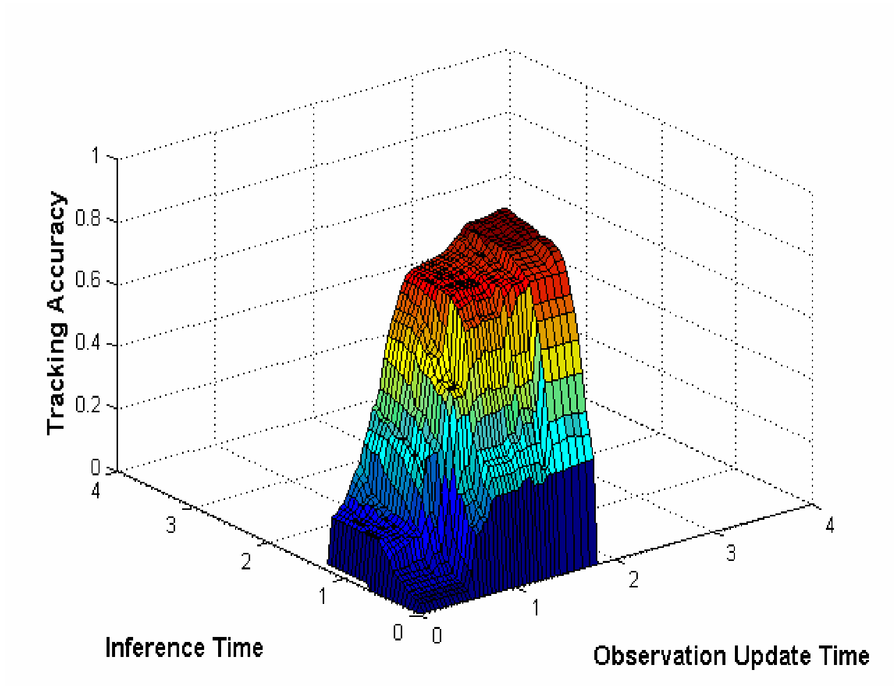
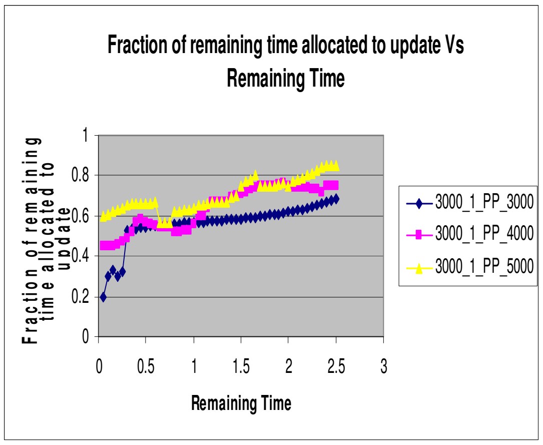


Figure 46: Performance profile for dataset 3000_1



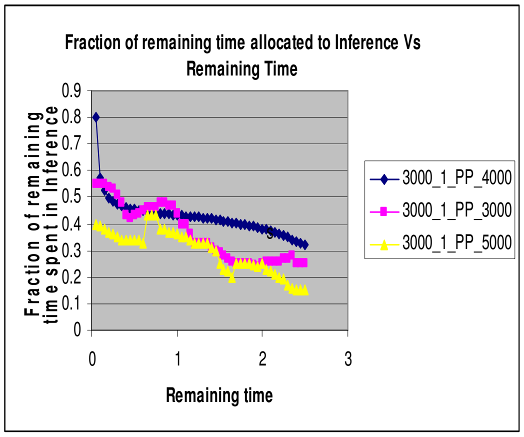


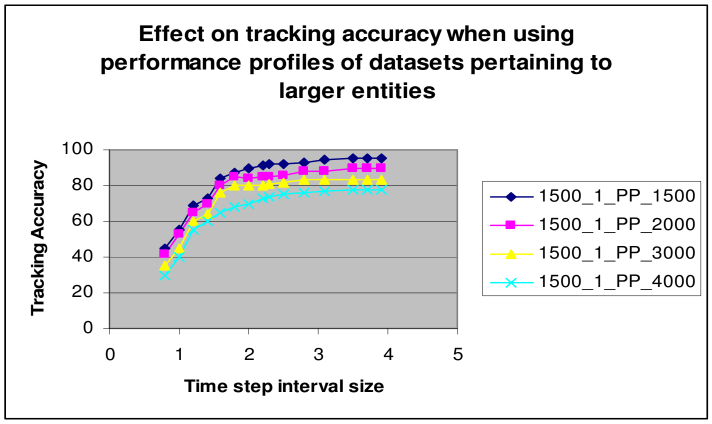
Figure 47. This figure shows the fraction of remaining time that gets allocated to the Observation Update task when the numbers are derived from the models pertaining to datasets 3000_1, 4000_1 and 5000_1

Figure 48. This figure shows the fraction of remaining time that gets allocated to the Inference task when the numbers are derived from the models pertaining to datasets 3000_1,4000_1 and 5000_1

Figures 44, 45 and 46 are the graphical representations of the performance profiles pertaining to datasets 5000_1, 4000_1 and 3000_1 respectively. Models for optimal composition of tasks are derived from these performance profiles in the range of time allocation units pertaining to the 3000_1 dataset.

Figures 47 and 48 show how the optimal compositions for Observation Update and Inference tasks vary for a given total time allocation when the models are derived from performance profiles pertaining to datasets 5000_1, 4000_1 and 3000_1 respectively.

We then perform experiments to study the effect on tracking accuracy for the 3000_1 dataset where the run time control uses the performance profile pertaining to the three different datasets mentioned above. Figure 48 illustrates the results so obtained. We perform a similar set of experiments on the data set 1500_1 using performance profiles pertaining to datasets 1500_1, 2000_1,3000_1 and 4000_1.



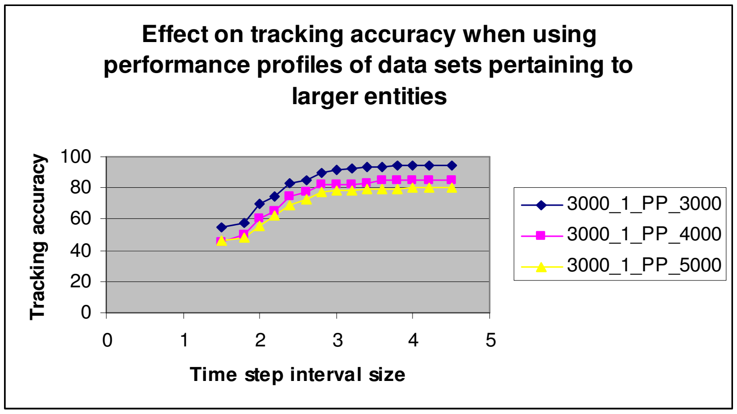


Figure 49. This figure illustrates the effect of tracking accuracy when the composition of tasks is based on performance profile models pertaining to another data set with higher number of entities. The legend entry 3000_1_PP_3000 signifies that the performance profile for the 3000_1 dataset is applied to the same dataset. 3000_1_PP_4000 signifies that the performance profile for the 4000_1 data set is applied to the 3000_1 dataset and so on.

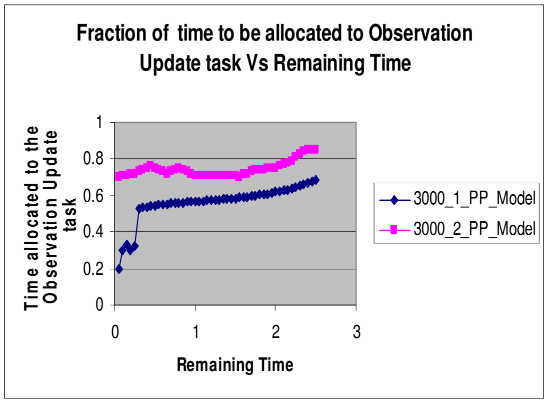
In both the cases, we make the following observations. There is a significant difference between the models obtained for datasets having different number of entities. This variance becomes more significant as the difference between the number of entities grows i.e. the difference between models obtained for datasets with 3000 and 5000 entities is higher than that between 3000 and 4000 entities. The difference between tracking accuracies obtained on the dataset 3000_1 by using the performance profiles for 4000_1 and 5000_1 respectively is approximately 7% and 12% respectively. Similarly the difference between tracking accuracies obtained on the dataset 1500_1 by using performance profiles for 2000_1, 3000_1 and 4000_1 is on an average is 4.5 %, 10% and 13% respectively.

The conclusions we draw from this experiment is that the performance profiles for our reference implementation are sensitive to the number of entities in a data set .This leads to a potential need to learn these performance profiles on line. Instead of having to incur the run time cost of online learning, another approach to handle this sensitivity would be to have a library of "context specific" performance profiles pertaining to different number of entities (the context being the number of entities in the case of our system prototype) and then using these performance profiles to predict the allocation of resources for smaller number of entities (context specific performance profiles are a group of performance profiles derived as a function of some critical input data parameter to which the performance profile is found to be sensitive). This method might suffice and eliminate the need for online learning of performance profiles provided we have enough "context specific" performance profiles to circumvent the degradation in solution quality which we observe in our results. This is essentially a design decision that needs to be addressed .The choice between maintaining a library of "context specific" performance profiles and online learning of performance profiles has to be based on the trade-offs between the cost/benefits of learning and use "context specific" performance profiles.

7.4.3 Sensitivity to entity behaviors

In this section, we discuss the sensitivity of performance profiles to the behaviors associated with entity types in a dataset. We generate two new datasets (3000_2 and 5000_2) with a new set of behaviors for the various entity types. The behaviors associated with the entity types differ from the baseline behaviors in terms of start and end times and start and end positions on the scenario map (for more details please refer to Section 4.2).

Figure 50 shows the optimal allocation of time to Inference and Observation Update tasks based on models derived from the performance profiles pertaining to data sets 3000_1 and 3000_2. These two data sets differ in the set of behaviors associated with the different types of entity types. As evident from the figure, there is a significant difference between the two models. The effect on tracking accuracy of applying the performance profile of the dataset 3000_2 to the dataset 3000_1 is illustrated in Figure 51 and that of applying the performance profile of the dataset 5000_2 to the dataset 5000_1 is illustrated in Figure 52. The tracking accuracy degrades on an average by 13% across the two data sets when using the performance profile pertaining to a dataset with different behaviors.



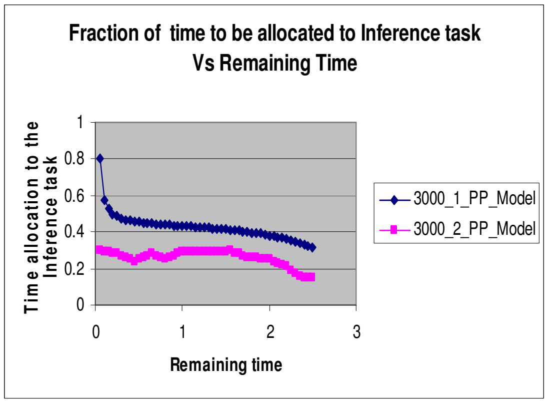


Figure 50 These figures show the fraction of remaining time that get allocated to the Observation and Inference tasks when the numbers are derived from the models pertaining to datasets 3000_1 and 3000_2 datasets

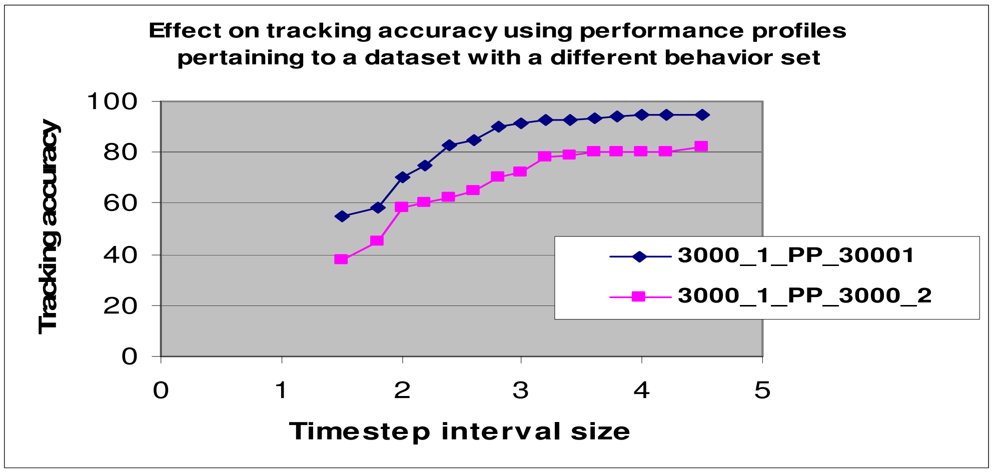


Figure 51. This figure illustrates the impact on tracking accuracy when applying the performance profile for a data set with a specific set of behaviors to another with a different set of behaviors. The legend entry 3000_1_PP_3000_1 signifies that the performance profile for the 3000_1 dataset is applied to the same dataset. 3000_1_PP_3000_2 signifies that the performance profile for the 3000_2 data set is applied to the 3000_1.

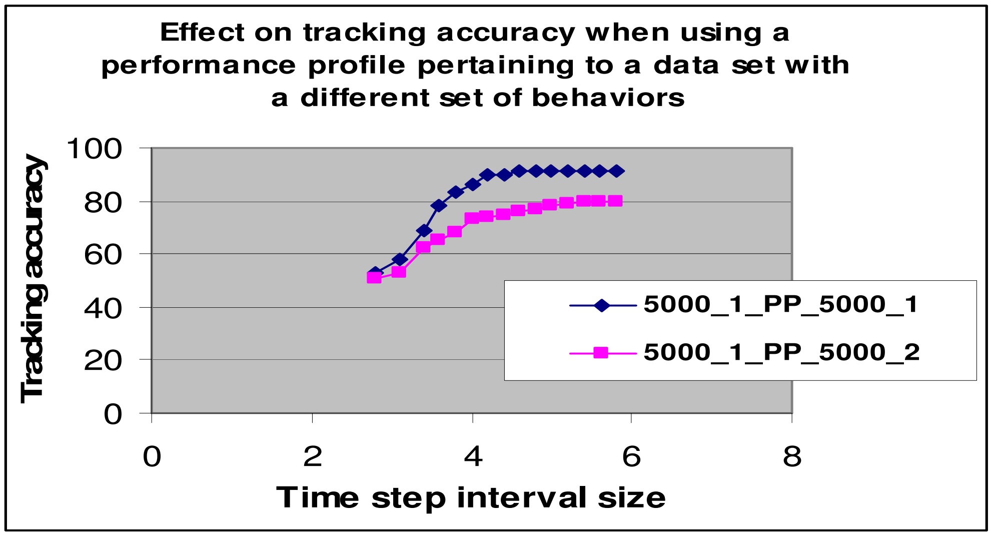


Figure 52. This figure illustrates the impact on tracking accuracy when applying the performance profile for a data set with a specific set of behaviors to another with a different set of behaviors. The legend entry 5000_1_PP_5000_1 signifies that the performance profile for the 5000_1 dataset is applied to the same dataset. 5000_1_PP_3000_2 signifies that the performance profile for the 5000_2 data set is applied to the 5000_1.

Though these are preliminary results, we tentatively conclude that the performance profiles are sensitive to the behavior parameters of the entities in a data set.

Sensitivity of performance profiles to different input data parameters opens up other important questions pertaining to research in the area of soft computing and it's applications to real time system design. One important issue is the online learning of performance profiles and the analysis of the cost-benefit trade-off of on line learning. Whenever we have a system in which performance profiles are sensitive to input data parameters, the first question that needs to be answered is whether maintaining "context specific" performance profiles (for critical input data parameter values) will suffice to meet the system objective. If not, one needs to look at ways to learn the performance profiles online. One of the main issues in on line learning is to ensure recognizability of the soft kernels, a desirable property of soft computations that we have not addressed in this thesis. As defined in Chapter 5, "recognizability" of a soft computation pertains to the ability of determining the impact of the computations on the system solution at run time and hence facilitates the learning of performance profiles online. Determination and evaluation of the degree of recognizability of the soft kernels in our system prototype is a part our planned future research in this area.

Chapter 8

8. Conclusion and Future Work

In this thesis, we address the design issues involved in exploiting "soft computing" properties of kernels to improve real time performance of systems. We build a system prototype of a real time scenario and application (real time sensor fusion and situational assessment) described in detail in Chapter 4. The development of this system prototype involved the porting of the MATLAB code of the Information Data Association algorithm to C, integrating and applying the algorithm to a cognitive test bed (simulates a surveillance and situational assessment scenario) provided by Northrop Grumman and developing a preliminary framework for investigating different real time scheduling policies. In the context of this system prototype and reference scenario, we demonstrate the benefits that soft computations provide to optimize real time system performance by enabling the meeting of system goals and deadlines with sufficient system solution quality without having to design the system to support worst case program execution. There are a number of conclusions that we draw from our work.

The first conclusion that we make is that identifying soft computations in applications and quantifying the gain in solution quality of the application and system as a function of time spent in soft computations (using individual and composite performance profiles) is an important first step to analyzing the opportunities that these computations provide to meet system goals with sufficient solution quality at a deadline.

Secondly, we propose a methodology of using models derived from composite performance profiles pertaining to two or more “soft computing” kernels to determine optimal allocation of resources/ time to these kernels at run time in order to be able to meet deadlines with sufficient system solution quality. We experimentally demonstrate that the use of these models by a dynamic scheduler provide on an average 39% improvement in system solution quality without missing any deadlines as compared to solution qualities obtained by using conventional real time scheduling techniques when designing the system for supporting less than worst case program execution. This improvement in solution quality is achievable while keeping the CPU highly utilized. We observed that with our dynamic scheduler, the tightest real time constraint that can be met on our reference CPU (3GHz Xeon) is on an average 1.65 times smaller than what can be achieved by a conventional real time scheduling scheme at comparable levels of functional correctness. Though the numbers by themselves cannot be considered representative of all real time systems and scenarios, we conclude that our scheme can provide significant benefits and facilitate real time system design at less aggressive design points than what is achievable through conventional real time scheduling and design techniques.

Thirdly, through a series of experiments, we identify the sensitivity of performance profiles to different input data parameters in the case of our reference real time scenario implementation. We conclude that in the context of our system prototype, assuming no constraints on input parameters, having a library of “context specific” or adaptive performance profiles and online learning of performance profiles has the potential of providing more performance benefits than achievable by our scheme, which uses a single performance profile determined off-line (prior to application execution), to make run time decisions about task compositions. However, using our scheme the improvements we have claimed earlier are achievable assuming some constraints in the problem formulation with reference to different input data parameters.

In a broader context, our work is a very preliminary step towards the realization of a new methodology for designing systems pertaining to certain real time application domains. The experimental results and analysis done as a part of this thesis, though preliminary and specific to our reference real time system scenario and prototype, point towards a possible new paradigm of real time systems design. The classical approach to the design of real time systems is to design a system with sufficient resources to meet worst case program execution given an application and a set of timing constraints. What our results demonstrate is that it might be possible to design a real time system to support less than worst case scenarios and still achieve the system goals predictably by exploiting soft computing properties of kernels using models derived from their composite performance profiles to dynamically schedule tasks at run time.

As a part of our future work in this area, we plan to address the design issues pertaining to the online learning of performance profiles. In addition to this, we plan to analyze the applicability of our methodology to other real time scenarios with more "soft computing" kernels. By doing so, we hope to be able to determine the feasibility of using our approach in the context of the design of real time systems in general.

Bibliography

1. C. L. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment". Journal of the ACM, 20(1):46--61, Jan. 1973.

2. Kang G. Shin and P.Ramanathan, "Real-time computing: A new discipline of computer science and engineering". Proc. IEEE, vol.82, no.1, pp.6-24, Jan. 1994.

3. Stankovic J. A., "Misconceptions about real-time computing: a serious problem for the next-generation systems". IEEE Computer., V21 N10, Oct. 1988, pp 10-19.

4. K. Ramamritham and J. Stankovic. "Scheduling algorithms and operating systems support for real-time systems". Proceedings of the IEEE, 82(1):55-67, 1994.

5. N. Audsley, A. Burns. "Real-Time System Scheduling". Technical Report YCS 134, University of York, UK.

6. F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-Vincentelli. "Scheduling of embedded real-time systems." IEEE Design and Test of Computers, 1998.

7. J. A. Stankovic, M. Sprui, M. DiNatale, and G. C. Buttazzo. "Implications of classical scheduling results for real-time systems." IEEE Computer, 28:16--25, June 1995.

8. D.B. Stewart and P.K. Khosla, "Real-Time Scheduling of Sensor-Based Control Systems," in Real-Time Programming (W. Halang and K. Ramamritham, eds.), Tarrytown, NY: Pergamon Press, 1992.

9. D. Sero, et. al., "On Task Schedulability in Real-Time Control Systems", 1EEE RTSS, December 1996.

10. W. Zhao, K. Ramamritham and J. Stankovic, "Preemptive Scheduling Under Time and Resource Constraints," IEEE Trans. on Computers, Vol. C-36,No.8

11. J. Stankovic and K. Ramamrithm. "The Spring Kernel: A New Paradigm for Real-Time Systems." IEEE Software, 8(3), May 1991.

12. G. Beccari, et. al., "Rate Modulation of Soft Real-Time Tasks in Autonomous Robot Control Systems," EuroMicro Conference on Real-Time Systems, June 1999.

13. S. Baruah and J. Haritsa. "Scheduling for overload in realtime systems". IEEE Trans. on Computers, 46(9), Sep 1997.

14. G. Buttazzo, and John A. Stankovic, "RED: Robust Earliest Deadline scheduling," in Proc. 3rd Intl. Workshop on Responsive Computing Systems, pp. 100-111, Sep. 1993.

15. K. Jeay, D.F. Stanat, and C.U. Martel. "On non-preemptive scheduling of periodic and sporadic tasks". In Proc. of the Twelfth IEEE Real-Time Systems Symposium, pages 129--139. IEEE Computer Society Press, 1991.

16. Winming Li,Krishna Kavi and Robert Akl."A Non-preemptive scheduling algorithm for Soft Real-time Systems".In Proc. of the Twelfth IEEE Real-Time Systems Symposium, pages 129--139. IEEE Computer Society Press,1991.

17. Wei Kuan Shih, J. W.-S. Liu, and C. L. Liu. "Modified rate-monotonic algorithm for scheduling periodic jobs with deferred deadlines". IEEE Transactions on Software Engineering, 19(12):1171--1179, December 1993.

18. Mahmoud Naghibzadeh." A modified Version of Rate-Monotonic Scheduling Algorithm and its efficiency Assessment, Seventh IEEE International Workshop on Object-Oriented Real-time Dependable Systems". San Diego, USA, January 7-9, 2002.

19. Peng Li, Binoy RAvindran,"Fast,best effort real time scheduling algorithms",IEEE Transactions on Software Engineering,19(12):1171--1179, December 2004.

20. M. Spuri and G. Buttazzo, "Scheduling Aperiodic Tasks in Dynamic Priority Systems", Real-Time Systems, vol. 10, pp. 179--210, 1996.

21. Vahid Salmani, Saman Taghavi Zargar, and Mahmoud Naghibzadeh,"A Modified Maximum Urgency First Scheduling Algorithm for Real-Time Tasks",Transactions on Engineering, Computing and Technology V9 November 2005 ISSN 1305-5313

22. D.B. Stewart and P. Khosla, "Real-Time Scheduling of Dynamically Reconfigurable Systems," IEEE International Conference on Systems Engineering, August, 1991, pp. 139-142.

23. Stankovic, JA., "Distributed Real-time Computing: The Next Generation," Jnl. of the Society of Instrument and Control Engineers of Japan, 1992.

24. J.W.S. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung, "Imprecise Computations," Proc. IEEE, vol. 82, Jan. 1994.

25. J. W.-S. Liu, K.-J. Lin, W.-K. Shih, A. C.-S. Yu, C. Chung, J. Yao, and W. Zhao, "Algorithms for scheduling imprecise computations". IEEE Computer, 24(5):58--68, 1991.

26. C. McElhone, "Soft Computations within Integrated Avionics Systems," in Proceedings of the IEEE National Aerospace and Electronics Conference (NAECON 2000.

27. Chen, I.R., "On Applying Imprecise Computation To Real-Time AI Systems," The Computer Journal, vol. 38, no.6, 1995.

28. S. Baruah and J. Haritsa. Scheduling for overload in realtime systems. IEEE Trans. on Computers, 46(9), Sep 1997.

29. A. Burns and D. Prasad, "Value-Based Scheduling of Flexible Real-Time Systems for Intelligent Autonomous Vehicle Control", Proceedings of the 3rd. IFAC Symposium on Intelligent Autonomous VehiclesMarch 1998.

31. T. Schwarzfischer, "Using Value Dependencies to Schedule Complex Soft-Real-Time Applications with Precedence Constraints". Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA), Nottingham (England), August 2003, ISBN: 0-9545821-0-1

32. F. Daoud,"Coordination factors in adaptive scheduling for soft real-time cooperative tasks: a performance
study",1996 IEEE International Joint Symposia on Intelligence and
Systems (IJSIS '96)

33. J. R. Haritsa, M. Livny, and M. J. Carey. Earliest deadline scheduling for real-time database systems. In Proc. of Real-Time Systems Symposium, pages 232--242, 1991.

34. S. K. Baruah and M. E. Hickey. Competitive on-line scheduling of imprecise computations. IEEE Transactions on Computers, 47(7): 1027-1033, September 1998.

35. C. McElhone, "Adapting and Evaluating Algorithms for Dynamic Schedulability Testing," Tech. Rep. YCS 225, Department of Computer Science, University of York, England, 1994.

36. O. Gonz alez, H. Shrikumar, J. A. Stankovic, and K. Ramamritham. Adaptive fault tolerance and graceful degradation under dynamic hard real-time scheduling. In Proceedings of the Eighteenth Real-Time Systems Symposium, pages 79--89, Dec. 1997.

37. C. McElhone and A. Burns. Scheduling optional computations for adaptive real-time systems. Journal of Systems Architectures, 46:46--77, 2000.

38. J.-Y. Chun, J. W.-S. Liu, and K.-J. Lin. Scheduling periodic jobs that allow imprecise results. IEEE Trans. on Computers, 19(9):1156--1173, 1990.

39. Stankovic, J., K. Ramamritham, and D. Niehaus, "On Using the Spring Kernel to Support Real-Time AI Applications," Proc. Euromicro Workshop on Real-Time Systems, June 1989.

40. Stankovic, JA., "Distributed Real-time Computing: The Next Generation," Jnl. of the Society of Instrument and Control Engineers of Japan, 1992.

41. D. J. Musliner, et al. The Challenge of Real-TIme AI. Computer (January): 58-66, 1995.

42. T. Schwarzfischer: Quality and Utility - Towards a Generalization of Deadline and Anytime Scheduling. Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS), Trento (Italy), June 2003, ISBN: 1-57735-187-8

43. Chen, I.R., "On Applying Imprecise Computation To Real-Time AI Systems," The Computer Journal, vol. 38, no.6, 1995.

44. D. W. McMichael, "Data fusion for vehicleborne mine detection," in MD98 [24], pp. 167--171.

45. M. Factor. Real-Time Data Fusion in the Intensive Care Unit. IEEE Computer, 1991, 24(11) : 45-54.

46. Will Meilander, Johnnie Baker, and Mingxian Jin,"Predictable Real-Time Scheduling for Air Traffic Control", in Proc. of the 15th International Conference of Systems Engineering, pages 533-539, August 2002

47. Xuanhua Li and Donald Yeung,"Exploiting Soft Computing for Increased Fault Tolerance,Appears in Workshop on Architectural Support for Gigascale Integration", Boston, MA. June
2006.

48. P. Dubey, “Recognition, Mining and Synthesis Moves Computers to the Era of Tera,” Technology @ Intel Magazine, pp. 1–10,February 2005.

49. M. A. Breuer, “Multi-media Applications and Imprecise Computation,” in Proceedings of the 8th Euromicro Conference on Digital System Design, pp. 2–7, September 2005.

50. S. Zilberstein. ,"Operational Rationality through Compilation of Anytime Algorithms",Ph.D. dissertation, Computer Science Division, University of California at Berkeley, 1993.

51. S.J. Russell and S. Zilberstein,"Composing Real time systems", Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, pp. 212-217, Sydney, Australia, 1991

52. S. Zilberstein,, "Using Anytime Algorithms in Intelligent Systems", AI Magazine, 17(3):73-83, 1996.

53. Jane W. S. Liu,"Real-Time Systems",Prentice Hall Inc,2000

54. Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. “Priority inheritance protocols: An approach to real-time synchronization.” IEEE Transactionson Computers, 39(9):1175– 1185, September 1990.

55.Chris Archer(Northrop Grumman), “Distributed Sensing Cognitive test bed conceptual design”, January, 2005 .

56. Chris Archer(Northrop Grumman), ”Distributed UAV Sensing Challenge Problem Test Bed Theoretical Basis Document”, Version 0.3 for the Cognitive Enabled Architectures Program under the DARPA Architectures for Cognitive Information Processing. January, 2004.

57. Stewart Frederick Edgar, "Estimation of Worst Case Execution Time Using Statistical Analysis", Ph.D Dissertation, University of York Department of Computer Science September 2002

58. Brad Schumitsch, Sebastian Thrun, Gary Bradski and Kunle Olukotun, “The Information-Form Data Association Filter.” Advances in Neural Information Processing Systems 18, Page 1193-1200, 2005

59. Y. Bar-Shalom and X.-R. Li, “Estimation and Tracking: Principles, Techniques, and Software.” YBS, Danvers, MA, 1998.

60. D.B. Reid., ”An algorithm for tracking multiple targets. IEEE Transactions on Aerospace and Electronic Systems,” AC-24:843–854, 1979.

61. Pang-Ning Tan, Michael Steinbach, and Vipin Kumar, ”Introduction to Data Mining,” Pearson, Addison Wesley, 2006.