ADAPTIVE TRACKING USING APPLICATION-LEVEL QoS (U)

Raymond K. Clark and Yun Zhang

(U) Abstract

(U) This paper describes an advanced technology demonstration (ATD) that applied utility-based scheduling to produce an adaptive, distributed tracking component appropriate for consideration by the Airborne Warning and Control System (AWACS) program. This tracker was designed to evaluate application-specific Quality of Service (QoS) metrics to quantify its tracking services in a dynamic environment and to derive scheduling parameters directly from these QoS metrics to control tracker behavior. The prototype tracker was initially implemented on the MK7 operating system, which provided native utility-based processor scheduling. More recently, it has been implemented as a real-time Java (i.e., RTSJ) application. In each implementation, the prototype updates all of the tracked-object records when the system is not overloaded, and gracefully degrades when it is. The prototype has performed extremely well during demonstrations to AWACS operators and tracking system designers. Quantitative results are presented for both the MK7 and RTSJ implementations.

I. (U) Introduction

(U) Many currently deployed computer systems are insufficiently adaptive for the dynamic environments in which they operate. For instance, the AWACS Airborne Operational Control Program, which includes the tracking system, has a specified maximum track-processing capacity. If the tracker processes data in FIFO order, it fails to process later data if its processing capacity is exceeded. Since sensor reports generally come in the same order from one sweep to the next, it is likely that, under overload, sensor reports from a specific region will not be processed for several consecutive sweeps. This is a potentially serious problem because there is no inherent correlation between important regions of the sky and the arrival order of sensor reports.

(U) This situation, while undesirable, is handled by skilled operators, who recognize that some data are not being processed and take remedial actions (e.g., reducing the gain of a sensor or designating portions of the sky that do not need to be processed). While these manual adaptations reduce the tracker workload, they are not ideal. Reducing sensor gain might cause smaller, threatening objects to go undetected.

(U) Injecting more intelligence into the tracker could avoid such compromises. The US Air Force Research Laboratory at Rome (NY), The MITRE Corporation, and The Open Group Research Institute undertook a joint project to explore that approach. Among other things, the project produced an Advanced Technology Demonstration (ATD) featuring a notional, adaptive AWACS tracker.

(U) The ATD tracker "processes the right tracks at the right time" by appropriately managing the resources needed for track processing. The prototype updates all tracked-object records when it has sufficient resources, and gracefully degrades otherwise. A single underlying mechanism automatically provides this degradation, despite its manifestation as a succession of qualitative operational changes: First, more important tracks receive better service than less important tracks while all tracks continue to be maintained. Under severe, sustained, resource shortages less important tracks are lost before more important tracks. Moreover, the tracker automatically delivers improved service whenever more resources become available—in essence, tracker performance gracefully degrades and gracefully improves without direct human intervention.

II. (U) Adaptivity, Utility-Based Scheduling, and Quality of Service

(U) Military planners have observed that "you never fly the same mission twice," implying that flexibility and evolvability are important design characteristics. With that in mind, we employed a relatively straightforward approach to adaptivity for this project—decomposing an application (or a set of applications) into component computations, which are then assigned application-specific utilities reflecting their individual contributions to the overall mission. By scheduling computations to maximize the accrued application-specific utility, the overall system can perform a given mission well.

(U) While application decomposition with appropriate utility assignments assists in effectively accomplishing a specified mission, application effectiveness could be further increased by employing feedback to directly

R. K. Clark and Y. Zhang are with The MITRE Corporation, Bedford, MA 01730. E-mail: {rkc, yzhang}@mitre.org.

drive its behavior. To do this, application-specific figures of merit (that we refer to as Quality-of-Service (QoS) metrics) are specified at design-time, and evaluated or estimated dynamically at run-time in order to monitor—and subsequently control—overall application operation. Section IV discusses this in some depth for the AWACS ATD, where the adaptive tracker uses feedback based on QoS metrics for individual tracks to determine the allocation of processing resources for current track processing.

III. (U) The Tracking Problem

(U) Surveillance radar systems are an important class of real-time systems that have both civilian and military uses. These systems consist of components for sensor processing, tracking, and display. In this study we concentrated on the tracking component, which receives *sensor reports*—the output produced by the sensor-processing component—and uses them to detect objects and their movements [1]. Each object is typically represented by a *track record*. Sensor reports arrive at a tracker periodically, and each report describes a potential airborne object. The number of sensor reports can vary from radar sweep to radar sweep, and some sensor reports represent noise or clutter rather than planes or missiles. When new sensor reports arrive, a tracker correlates the information contained in the sensor reports with the current estimated track state to update the track records that represent the tracking system's estimate of the state of the airspace.

(U) A typical tracker comprises *gating*, *clustering*, *data association*, and *prediction and smoothing* stages. Gating and clustering splits (*gates*) the problem data into mutually exclusive, collectively exhaustive subsets of sensor reports and track records, called *clusters*. Data association then matches the cluster's sensor reports with its track records. The final stage of a tracker—prediction and smoothing—computes the next position, velocity and other parameters for each object using its track history and the results of data association.

A. (U) Adaptive Tracking

(U) The motivating problem was the (mis)behavior of the tracker under overload, which can be intentionally caused by an enemy. Since, under overload, all of the incoming data cannot be processed, the project's goal was to allow the tracker to do a better job of selecting a subset of incoming data that could feasibly be processed by allowing scheduling decisions that had previously been made at design time to be deferred until run time. There were two major changes: processing was divided into smaller-grained units of work that could be scheduled independently and concurrently; and utility-based design principles were employed to determine appropriate scheduling parameters for those individual work units.

(U) Given a multithreaded design, with a separate thread assigned to perform each association computation, and an underlying scheduler that attempts to maximize accrued utility, the adaptive tracker design problem is reduced to a straightforward question: Can scheduling parameters be selected for tracks and clusters that reflect the utility associated with their processing?

(U) Considerable effort was devoted to answering that question for the ATD. Fortunately, at about this time, the tracking community was independently encouraged to think in terms of "selling" track information to customers—that is, operators and decision makers. That point of view helped make the notion of establishing the application-specific utility of a track quite natural. In fact, the project developed a set of QoS metrics for individual tracks. These per-track QoS metrics, described in Section IV-B, directly determine the scheduling parameters for a cluster, which, in turn, affect the future QoS metrics of each component track.

IV. (U) Utility-Based Design

(U) AWACS can perform a number of different missions: e.g., it can manage logistics such as refueling, perform air-traffic control, or carry out general surveillance. In order to maximize the depth of this initial work, we applied the principles of utility-based design to a single mission. Because of its general usefulness and intuitive simplicity, a *surveillance* mission was chosen for the ATD.

(U) When flying a surveillance mission, AWACS operators attempt to monitor all airborne objects in a large region. Once an object has been identified, the tracker follows its progress (i.e., "tracks" it). The more closely the tracker's estimate of an object's position and heading agree with reality, the better.

(U) Moreover, once the tracker has identified a track, it should not "drop" it erroneously. A track is dropped if it is not updated for a number of input sensor cycles. While this is inevitable if no new sensor input is received for the track, the project focused on cases where sensor input is received, but is not processed.

A. (U) Adaptive Tracker Behaviors

(U) A key step to producing an adaptive tracker was to develop a specification of its desired behavior—in particular, its behavior under overload. This specification occurred in two phases. The first phase described a high-level policy, but did not address tradeoffs that could arise when attempting to satisfy conflicting policy objectives. For example, the high-level adaptive tracker should preferentially process tracks that (a) are in danger of being dropped, (b) the user has identified as "important," (c) have poor state (position and velocity) estimates, (d) are maneuvering, (e) potentially pose a high threat, or (f) are moving at high speed.

(U) The second phase refined this high-level policy by addressing conflicting policy objectives. In general, project members could identify these conflicts, but needed expert assistance to determine the proper resolutions (i.e., tradeoffs).

B. (U) QoS Metrics for Tracks

(U) Policy refinement required the definition of QoS metrics. Where the high-level policy could usefully describe behaviors for more and less important tracks, or make a distinction between more and less accurate track positions, the implementation of that policy required precise specification of these terms. The project settled on three QoS metrics that could quantify track processing:

- (U) Timeliness: the total elapsed time between the arrival of a sensor report and the update of the corresponding track record.
- (U) Track quality (TQ): a traditional measure of the amount of recent sensor data incorporated in the current track record. TQ is incremented or decremented after each scan and ranges between zero (lowest quality) and seven (highest quality). If TQ falls to zero, the track is dropped.
- (U) Track accuracy: a measure of the uncertainty of the estimate of the track's position and velocity.

(U) In addition to these QoS metrics, each track was dynamically assigned to one of two *importance classes*. A track could be deemed more important for a number of reasons, including designation by an operator, flying in an operator-designated region, or posing a significant threat to the AWACS platform.

C. (U) Quantified Adaptations

(U) Based on these classes, project members refined the high-level adaptive tracking policy by focusing on specific tradeoffs involving pairs of track QoS metrics or track importance. Several such pair-wise rankings were merged to form a total ordering of the cases (with numeric values) as a function of track quality, track accuracy, and track importance.

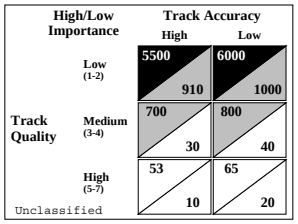

Fig. 1. (U) Track Utilities as a Function of Track QoS Metrics

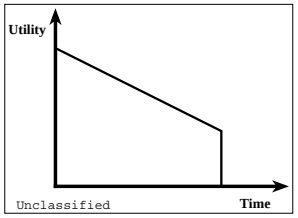

Fig. 2. (U) Utility Functions for Association Computations

(U) At that point, tracking experts helped quantify the relative utilities of the bins. We also examined several cluster scenarios "by hand" to assign overall utilities and perform fine tuning. Figure 1 shows the set of valuations ("bins") that were employed for our demonstrations, which are described briefly in Section VIII.

(U) Once the bins had been assigned utilities, the translation to utility-based scheduling parameters was straightforward. Each time constraint describes the application-specific utility (in this case, the bin value of the cluster) of completing the designated computation as a function of time. For association computations, all of the time constraints had a similar shape (see Figure 2), which specified that "sooner was better than later."

V. (U) Prototype Implementations

(U) Several prototype implementations have been produced to date. The initial implementation was based on The Open Group Research Institute's MK7 operating system [6][7], which offered native support for utility-based scheduling. That implementation performed well; results are presented in Section VIII.

(U) More recently, the Real-Time Specification for Java (RTSJ) [2] has been undertaken under the auspices of the Java Community Process. It brings real-time capabilities to the Java programming environment, including selected provisions for advanced scheduling policies. While no implementation of utility-based scheduling has appeared to this point, such a scheduling policy can be hypothesized. In anticipation of this possibility—and to take advantage of the ubiquity and portability of Java—an RTSJ implementation of the adaptive tracker was produced.

(U) Two of the prototype implementations are described briefly in the following sections. Results produced by the prototypes are described in Section VIII.

VI. (U) The MK7 Prototype

(U) The adaptive ATD tracker builds on several features of the underlying operating system, The Open Group Research Institute's MK7 [6][7], which provides a number of standards-based facilities as well as a set of unique capabilities designed to support distributed, real-time applications.

(U) Beyond traditional real-time support (e.g., predictable execution times, preemption, priority scheduling, instrumentation, and low interrupt latency), MK7 provides a number of advanced features. The MK7 microkernel, which has been explicitly developed to support real-time applications, contains a scheduling framework that simultaneously supports priority-based and time-based scheduling policies. A utility-based processor-scheduling policy called Best-Effort scheduling [5][7], which accepts application (and system) time constraints and schedules computations according to a heuristic that attempts to maximize total accrued utility, was particularly useful for the ATD. Notably, this utility-based scheduling policy and the threads it schedules co-exist with and interact with other parts of the OS and application, including synchronizers, preemptions, and "priority" modifications (e.g., priority inheritances and priority depressions).

(U) MK7 threads are *distributed* (or migrating) threads[3][4], which move among the processes (i.e., MK *tasks*) of a distributed system by executing RPCs while carrying an environment that includes information like the thread's scheduling parameters, identity, and security credentials.

(U) MK7 provides several standard interfaces—including a highly standards-compliant UNIX interface and the X Window System—that permit application programmers to reuse existing code. Both distributed threads and utility-based scheduling are provided as generalizations or extensions of existing POSIX thread functions—thus simplifying the programmer's task considerably.

VII. (U) The RTSJ Prototype

(U) The Real-Time Specification for Java (RTSJ) [2] describes the requirements placed on conforming real-time platforms. The adaptive tracker offered a meaningful proof-of-concept application demonstration for RTSJ platforms.

(U) The MK7 prototype is written in C and C++. The RTSJ prototype was created by a two-step process. First, the tracker was reimplemented in (generic) Java. Then the prototype was modified to utilize critical RTSJ features. For instance, the RTSJ tracker uses NoHeapRealtimeThreads (with priorities ranging from 1 to 29). In addition, it imitates utility-based scheduling through the use of a dynamic, priority-based policy. (Arguably, this approach is not generally applicable; however, due to the nature of this specific workload, the approach does apply in this case.)

(U) The RTSJ platform selected was based on QNX RTP (real-time platform). It hosted the J9 Java virtual machine (JVM), the beta Real-time Extensions package, and the beta Personal Configuration package from IBM's subsidiary Object Technology International [8]. The Real-time Extensions package provides the initial implementation of portions of the RTSJ for the J9 JVM. The Personal Configuration provides Java remote method invocation (RMI) support that's used by the tracker. These packages were released through IBM's VisualAge Micro Edition product [9].

(U) In addition, a new component, called the Mediator, was introduced to act as a conduit to pass the sensor data through Java RMI to the tracker under QNX RTP.

VIII. (U) Prototype Results

(U) The MK7 prototype performed very well during demonstrations to its target audience—AWACS operators and tracking system designers—most notably at the Boeing AWACS Prototype Demonstration Facility. Under "normal" (non-overload) conditions, the tracker handled all tracks as expected and delivered high QoS for all tracks. Overload conditions were simulated by artificially tightening the deadlines for the completion of association processing.

(U) Figure 3 shows results of the MK7 tracker for a typical overload scenario. Batches of between ten and 14 sensor reports arrived at the tracker, with one-third of the tracks belonging to the more-important track class. The Association Capacity axis indicates the number of associations the tracker could usually perform in the specified processing time limit, and the Track Quality axis indicates the average TQ delivered for each track-importance class.

(U) Figure 3 indicates that when the MK7 tracker can only process about 33% of the input, the prototype delivered essentially perfect TQ for the more impor-

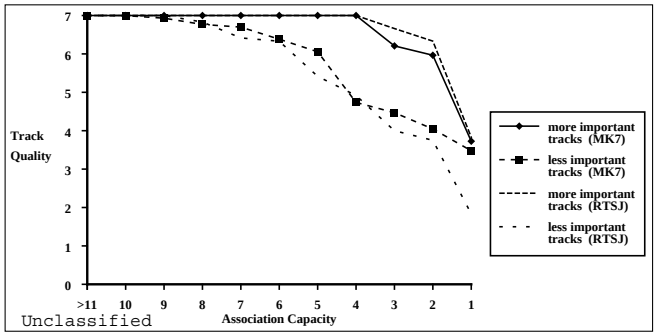

Fig. 3. (U) Average Track Quality as a Function of Association Capacity

tant tracks, while delivering a reasonable (about 4.5) TQ for the less important tracks. When the tracker was further constrained so that it could only process about 10% of the input, the prototype finally dropped some tracks—from the less important track class. No important tracks have been dropped during our demonstrations.

(U) In addition, the demonstrations have shown that the tracker also adapts when new resources are added. In that case, which we demonstrate by loosening the time constraints on association processing, the prototype automatically delivers approximately the maximum achievable QoS.

(U) The RTSJ tracker displays behavior similar to the MK7 tracker, as shown in Figure 3. On average, the two prototypes show essentially the same statistics at a given association capacity. Examination of actual associations performed at a given capacity level show slightly more variation under the RTSJ prototype than under the MK7 prototype.

IX. (U) Summary

(U) The AWACS ATD project produced an adaptive, distributed tracker that was directly driven by Quality-of-Service metrics. Based on a novel design and incorporating knowledge from experts in the field, this tracker gracefully handles overloads, addressing a problem with currently deployed trackers and with trackers under development. The tracker was demonstrated to AWACS operators and tracker designers at Boeing in September 1998 and received supportive feedback, particularly regarding its behavior under overload and the operator interface.

(U) This project provides another worked example in the area of utility-based scheduling and further encourages our confidence in this technology. The derivation of the QoS metrics for tracks provided valuable insight into the nature and use of application-specific QoS metrics in a new application domain. Finally, prototype trackers have been constructed that can be used for further experimentation and can host future extensions. The MK prototype employed native support for utility-based scheduling, while the RTSJ prototype imitated such scheduling based on a commercial product that supports the initial draft of the RTSJ. The RTSJ prototype displays encouraging behaviors and provides initial evidence of the usefulness of RTSJ for this class of applications.

(U) Acknowledgments

(U) The RTSJ prototype and many results in this paper build on work previously performed by Pat Hurley, E. Douglas Jensen, Arkady Kanevsky, Tom Lawrence, John Maurer, Paul Wallace, Douglas Wells, and Thomas Wheeler.

References

[1] Blackman, S.: Multiple-Target Tracking with Radar Applications. Artrech House, ISBN 0-89006-179-3 (1986)

[2] Bollella, G., Gosling, J., Brosgol, B., Dibble, P., Furr, S., Hardin, D., Turnbull, M.: The Real-Time Specification for Java. Addison-Wesley Publishing Company, ISBN 0-201-70323-8 (2000) [http://www.javaseries.com/rtj.pdf]

[3] Clark, R.K., Jensen, E.D., Reynolds, F.D.: An Architectural Overview of the Alpha Real-Time Distributed Kernel. Proc. of the USENIX Workshop on Micro-kernels and Other Kernel Architectures (1992) 127-146

[4] Ford, B., Lepreau, J.: Evolving Mach 3.0 to a Migrating Thread Model. Proc. of the USENIX Winter 1994 Technical Conference (1994)

[5] Locke, C.D.: Best-Effort Decision Making for Real-Time Scheduling. Ph.D. Thesis, Dept. of Electrical and Computer Engineering, Carnegie-Mellon University (1986)

[6] Wells, D.: A Trusted, Scalable, Real-Time Operating System Environment. Dual-Use Technologies and Applications Conference Proceedings (1994) II:262-270

[7] —: MK7.3 Release Notes. The Open Group Research Institute, Cambridge, MA (1997)

[8] —: Object Technology International [http://www.oti.com].

[9] —: IBM VisualAge Micro Edition. [http://www.embedded.oti.com].