Predictability for Real-Time Command and Control

Will C. Meilander, Johnnie W. Baker, Jerry Potter
Kent State University
willcm, jbaker, potter @mcs.kent.edu

Abstract

This paper describes a new and different paradigm for real-time command and control that we show can provide a static, polynomial-time scheduling algorithm. Current efforts in real-time scheduling have been unable to predict system performance, and use a unpredictable "dynamic" scheduling algorithm. The Nation's Air Traffic Control (ATC) System has been unable to satisfy performance requirements, and is extremely expensive. An editorial in "USA Today" (4-19-1999) cites expenditures in ATC at $41 billion. But, current multiprocessor technology cannot do the job.

The FAA wisely required a "proof of concept" study before the (AAS) contract. But that study, after expending a billion dollars, was abandoned, and production moved forward. AAS was canceled in 1995 exactly in line with real-time scheduling theory predictions. Garey, Graham, and Johnson state: "For these scheduling problems, no efficient optimization algorithm has yet been found, and indeed, none is expected."[9]. Stankovic et. al. state: "…complexity results show most real-time multiprocessing scheduling is NP-hard." [6]. We offer a completely different approach, that has been shown to overcome the severe limitations of multiprocessing. Additionally, we show that real-time parallel-processing techniques can be statically scheduled to solve the set of tasks making up the ATC problem for a worst-case environment.

Introduction

This paper considers scheduling and performance predictability, a subject of much interest in today's Command and Control (C&C) world. Predictability necessitates the ability to evaluate the implementation of a complete real-time system and predict the performance that can be achieved for some worst-case set of input parameters. We show numerous citations that acknowledge the impossibility for current mulitprocessor architectures to satisfy the problem of predictability for Command and Control Systems.

Current efforts in the field are committed to dynamic scheduling and indicate that predictability is a future goal. Hundreds of heuristic evaluations have been pursued in an attempt to provide predictable algorithms for dynamically scheduling multiprocessor (MP) systems in the real-time environment [25]. In this paper, we consider the question of whether or not a system using a dynamic scheduling algorithm can be predictable. It would seem that sufficient knowledge for predicting execution time cannot be provided if it becomes necessary to dynamically schedule tasks in a real-time multiprocessor (MP) or distributed processor environment. In particular, the need for dynamic scheduling acknowledges the inability to statically schedule, and thus the inability to predict the performance that can be realized with the scheduling algorithm. Torngren states that "…some researchers in control theory have drawn the conclusion that *computer systems are inherently probabilistic in terms of their timing behavior"* [19]. Torngren also quotes Halang who wrote that "…the use of contemporary computers inhibits real-time control."

An associative processor (AP) is a parallel SIMD architecture with some additional properties [11,22,23]. In particular, it provides hardware support for the set-oriented processing operators for relational databases, including executing queries, processing results of associative searches and global reduction operators (e.g., AND, OR, Maximum, and Minimum). Since SIMD parallel processors are synchronous and have only one instruction stream, they share the same ease of predictability as a simple conventional processor, as shown in Potter and Meilander [8]. In particular, the massive data parallelism of a single instruction parallel processor such as a SIMD or an associative processor (AP) can support static scheduling algorithms that can satisfy the real-time needs of C&C problems. Additionally, we show that APs can support C&C schedules that are fully deterministic and predictable.

A Database System

We first discuss the nature of C&C problems. The Nation's Air Traffic Control (ATC) system is an example of C&C systems. The ATC system is fundamentally a relational database system since it is readily configured as a set of relational tables, and thus provides a solid mathematical foundation for C&C processing. According to C. J. Date, "A database system is essentially nothing more than a *computerized record keeping system"* [1]. Most operations in not only the ATC problem but also in other C&C problems involve accessing and processing these computerized records. Data base systems currently play a critical role in most large real-time systems.

A relational database is fundamentally a set structure. The ATC database is no exception. Since this is true, we can assert that all operations that are performed on the database are set operations. A set operation is an operation that involves many elements of the database in a single job. Examples are:

1. Insert one or more new data elements into the database
2. Read one or more elements of data from the database
3. Modify one or more elements of data in the database
4. Delete one or more elements of data from the database.

It will be readily observed that each operation starts with a search, which is a set operation.

1. Search the database to find the proper place to insert new data.
2. Search the database to locate the items to be read
3. Search the database to select the items to be modified
4. Search the database to find the items to delete.

These search operations are inevitable, omnipresent, and cannot be avoided. These kinds of search operations exist in all database systems. The cashier at the grocery checkout counter searches for the price of an item the customer has purchased. The cashier has a bar-code reader that simplifies the task, but the database must be searched for the barcode value in order to find the price of the item. The cashier's problem may be considered "soft" real-time processing. In this context, “soft” means no damage will be done if the search is delayed for a time while other searches are processed.

On the other hand the ATC system is a "hard" real-time system. If data processing is not completed on time, undesirable things can happen. Two aircraft may collide or an aircraft may fly into a mountain. Alternatively, the controllers' screens may become blank, stopping the display of much needed information and allowing a runway incursion. All of these things actually have happened because the computer system couldn't handle the data given to the processor [3].

The ATC System deliberately ignores (a process called data mosaicing or “selective rejection”) information critical to flight safety simply because it does not have the ability to process and present that information to the controller in a useful manner. A “mythical feeling” has persisted since the first ATC implementation using MPs that systems are getting faster, and the need to ignore information in ATC will soon disappear. Such has not been the case. The perfectly reasonable requirements of the Automated Air Traffic Control System (AAS) program were canceled when it became evident, after an expenditure of billions of dollars, that a reasonable implementation of the requirement was unachievable. This impossibility was quite in consonance with a multitude of theoretical studies of real-time scheduling which have declared, unequivocally, that problems of this kind are intractable when implemented in a multiprocessor. FAA is asking more of the current multiprocessor system than its architecture will ever be able to supply.

We do not intend, in any way, to be critical of the thousands of people, dedicated managers, and highly skilled analysts and programmers who have worked diligently to bring about a solution to this problem. They were faced with an impossible task. That task, to implement the ATC data processing system using a multiprocessor architecture, is still impossible.

Decision Support System

Recently, there has been much discussion of a Decision Support System (DSS) for ATC[1]. But, DSS has always been the purpose of the ATC system. ATC, by its very existence, is a DSS. The system should always provide maximum information for the support of the controller. Today it does not! Mid-air collisions have occurred because the ATC system throws away useful data. For one example, see [3]. Today's system lets the controller down. There is much more information available that the controller could beneficially use, but today's ATC automation system ignores it. Less than ten percent of

[1] The US ATC system is made up of two different control systems. These are the terminal and enroute systems. Essentially, the terminal system controls terminal traffic, and the enroute system or Center controls traffic between terminals.

the data available within an ATC Center is made available to the controller.

Why are these limitations permitted to exist? It happens because there is a fundamental limitation in the processing system. That limitation is memory to processor bandwidth. Gregory Pfister writes in the third from last paragraph of his book, “In search of Clusters” [14], “Once a processor is fast enough to continuously saturate a memory system, what difference does its internal architecture make? For computer performance in a wide and increasingly broad class of applications, ‘It’s the memory, stupid.”

Present ATC processing systems are unable to process the amount of data that should be made available to the controller. Therefore, good data is thrown out. It doesn't pass though the "data rejection filter". In addition, useful sensor information from terminal control systems within each center is never even sent to the center for processing and utilization. It cannot be processed in today's systems.

Real-time system scheduling

A single conventional computer can statically schedule a limited (i.e. small) aircraft environment. This was demonstrated in FAAs ARTS-3, an excellent example of simplicity for the small terminal environment. However, the number of planes that need to be handled today greatly exceed the capacity of a simple sequential computer. The solution is parallelism. There are two fundamentally different types of parallelism – MPs and SIMDs. An associative processor (AP) is a SIMD computer with some important additional features that are needed for ATC type problems [11,22,23]. As will be discussed later, the MP requirements results in NP-hard real-time scheduling problems and adding more computers in the MP approach can actually increase the execution time reducing throughput. The alternative AP approach can provide a greatly expanded computing capacity without adding more instruction streams, thus providing for a much larger air traffic environment while still maintaining the same predictable, static scheduling algorithm that works for a sequential computer.

Most of today’s C&C systems try to develop adequate functionality using a combination of dynamically scheduled, conventional, off the shelf processors. Instead, we will discuss how to solve the ATC problem using static scheduling with a single instruction parallel processor (SIPP)[2]. First, we go over the following assumptions for real-time performance that appeared in a seminal paper by Liu and Layland [4]. These assumptions are also considered in Stankovic et. al. [5]. We modify them slightly for the single instruction parallel processor (SIPP) architecture.

1. All hard tasks are periodic, aperiodic, or sporadic, and all are scheduled.
2. Tasks are ready to run at their scheduled times.
3. Deadline times are equal to scheduled task release times.
4. Jobs do not suspend themselves.
5. Tasks are independent and do not require synchronization between them, nor shared resources, nor relative dependencies or constraints on release times or completion times.
6. Processing is not preemptable at any point.
7. There are overhead costs for scheduling or interrupt handling.

In addition to the timing constraints above, the following constraints and requirements given in [5] are essential to assure performance (and thus also limit performance) for a (multi-instruction stream) multiprocessor system. In the following restatement of these requirements for a SIPP, it is evident that they are easily satisfied.

1. Resource constraints - A task may require access to certain resources other than the Instruction Processing unit, such as I/O buffers and databases.
2. Precedence relationships - We recognize the importance of precedence constraints by incorporating them into the static schedule.
3. Concurrency constraints - Tasks are not allowed concurrent access of resources.
4. Communication requirements, except I/O are virtually non-existent in the single instruction processing system.
5. Placement Constraints - When multiple instances of the same task are executed for fault tolerance, they should be executed on different processors.
6. Criticalness - All tasks are considered equally critical in these hard real-time systems.
7. Deadline time is the only performance metric in the systems under consideration. If a deadline is missed, the system has failed.

Dynamic Scheduling

We will now review a few of the well-known problems with dynamic scheduling [7]. There are several different types of on-line scheduling algorithms. One partition of

[2] A SIMD or an AP is designated a SIPP to distinguish its architecture from the MP which has many instruction processors, one in each of the off the shelf conventional processors that make up the MP.

scheduling algorithms is into preemptive and non-preemptive algorithms. Preemptive algorithms allow the scheduler to suspend any process with very little overhead and later resume from the point of suspension. Observe that preemptive algorithms assume the use of dynamic scheduling. On-line scheduling algorithms can be classified as either *static priority* or *dynamic priority*. The authors in [7] state that "…most realistic problems incorporating practical issues such as task blocking and transient overloads are NP-hard."

Stankovic [6] states that "A dynamic scheduling algorithm … has complete knowledge of currently active tasks, but new task activations, not known to the algorithm when it is scheduling the current set, may arrive. Therefore the schedule changes over time. For example … military command and control applications require dynamic scheduling." and, "in the presence of shared resources the complexity of feasibility analysis becomes exponential." In [21] the authors observe that "Parallelization gives a scheduler the flexibility to allocate more processors to a job whose deadline is near. Unfortunately, with this flexibility, some of the multiprocessor scheduling problems are very difficult. We prove the NP–hardness of scheduling parallelizable jobs where each job has a fixed priority."

Static Scheduling

Klein et. al. [7,pg.24]state that "One guiding principle in real-time system resource management is *predictability*, the ability to determine for a given set of tasks whether the system will be able to meet all the timing requirements of those tasks. Predictability calls for the development of scheduling models and analytic techniques to determine whether or not a real-time system can meet its timing requirements."

Based on the preceding considerations, we will develop a deterministic polynomial scheduling algorithm for the ATC system. This, of course, is in sharp contrast with all the current scheduling algorithms using MPs for real-time systems. Addition of such simple requirements as shared resources, priority management, or communications interactions render the MP version of this real-time scheduling problem intractable. However, this scheduling problem is tractable, as will be shown in this paper.

A Static Schedule

We use the earliest deadline first (EDF) scheduling algorithm in our approach. Jackson's Rule [24] states that "Any sequence is optimal that puts the jobs in order of non-decreasing deadlines." Additionally, Dertouzos [20] shows that "The EDF algorithm is optimal in that if there exists any algorithm that can build a valid (feasible) schedule on a single processor, then the EDF algorithm also builds a valid (feasible) schedule." In discussing real-time uni-processor applications, Stankovic, et. al. [5] state that "When preemption is not allowed, the scheduling problem is known to be NP-hard. However, if only *non-idling* schedulers are considered, the problem is again tractable. Namely, a scheduler is non-idling if it is not allowed to leave the processor idle whenever there are pending jobs."

A static schedule without preemption is the approach we follow in designing schedules for real-time command and control using a single instruction parallel processor. Let's examine the problem environment, and then examine timing within the system.

Example Environment

We postulate a control system environment consisting of the following:

Area of coverage

- A region the size of the "Cleveland Center" (160,000 square miles)

Number of Aircraft

- A large number (4,000 local and 6,000 adjacent system backup) of IFR and VFR aircraft in flight or expected in the near future.

Input data

- A multiplicity of sensors providing data (8,000 reports/sec.) to the control system.
- Flight plan information, scheduled plus in flight modifications.
- Weather information updates.
- Controller requests.

Output data

- Recommended conflict resolution maneuvers.
- Recommended terrain avoidance maneuvers.
- Recommended action for aircraft deviating from flight plan.
- Controller requested data.
- Forced display information to controller.
- Cockpit situation display information (to pilots).
- Terminal air traffic management optimization.
- Automatic voice advisory information to cockpit (for VFR pilots).
- Flight status to Airlines.
- Flight status to other control facilities.

Control flow and timing

For the worst case environment presented above, the following tasks are developed and scheduled. Each of the tasks has a worst-case solution time. The total time period available, a greater period or the time for each complete solution cycle, is 8 seconds. The periods in the table below are call lesser periods and are established to manage the dataset correctly. Sensor reports include ADS-B, GPS, and scanning primary and secondary surveillance radars.

	Function	Period
1.	Report correlation (4,000 reports/.)	0.5 second
2.	Track smoothing	0.5 second
3.	Track prediction	0.5 second
4.	Flight plan to track conformation	4 seconds
5.	Conflict detection (4000 flights)	4 seconds
6.	Conflict detection (strategic)	8 seconds
7.	Conflict resolution (5 per period)	4 seconds
8.	Terrain avoidance	4 seconds
9.	Final approach (100 runways)	4 seconds
10.	Controller display	1 second
11.	Cockpit display	8 seconds
12.	Automatic voice advisory (AVA)	8 seconds
13.	Aperiodic and sporadic requests(200)	1 second
14.	Error detection and correction	1 second

Using the set of worst case conditions established under the “Example Environment” above, we apply Liu and Layland’s “necessary-and-sufficient condition for task schedulability for independent processes, namely

$$\sum p_j / T_j \leq 1$$

where p_j is the processing time and T_j is the period of task j [6]. We then find the summation of tasks, over the greater period, to equal 0.528, which shows the set of processes to be feasibly schedulable.

We point out that these times are for the worst-case conditions, and the real time will usually be much smaller. This solution of the real-time scheduling problem leaves a comfortable adaptation margin for changes and the addition of new functions. (In 1997, E. E. Eddey and Meilander developed many of the algorithmic and timing details in an unpublished work.)

Single Instruction Parallel Processing

It is difficult to describe the compelling simplicity and the vast extent of the processing throughput that can be realized using the SIPP architecture. The SIPP architecture is simplicity. There are many processors (thousands) under the command of one instruction-processing unit (IPU). Each of the parallel processors (PEs) has a memory, a set of registers, and an arithmetic logic unit (ALU) and can simultaneously, with all PEs, access memory.

But, unlike conventional "off the shelf" processors, the PEs cannot interpret or decode instructions. They can only execute commands received from the IPU. Each instruction from the IPU then can be executed on thousands of operands.

It is true that in a multiprocessor or distributed system, many operands can be accessed concurrently, but this "benefit" comes at the large communication cost of "moving" data (and often instructions) from one processor to another. Pfister [14] acknowledges that each data item access requires about 2.7 instruction accesses. In addition, it must be remembered that memory access on a MP also has to contend with the problems associated with maintenance of sequential consistency, memory and cache concurrency, accessing distant memory locations, etc.

It is useful to consider memory access in light of the original wording of Amdahl’s law [18], which states

> "A fairly obvious conclusion which can be drawn at this point is that the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by sequential processing rates of very nearly the same magnitude."

While Amdahl's law is interpreted in different ways, very few interpret memory access as a sequential operation. Nevertheless, it must be recognized that memory access is a sequential operation, and limits "parallel processing rates." It is generally understood that a conventional processor must access each operand sequentially, and uses about 2.7 instruction memory accesses for each operand access. Seymour Cray realized this limitation of sequential memories. He designed the Cray 1 with a wide bandwidth memory of 1,024 bits. Today, others are recognizing this problem and are trying to overcome this bandwidth limitation with designs of 64, 128, and even 256 bit wide memories. That is, each access “gets” 256 bits in one operation. This leads in the direction of “...sequential processing rates of very nearly the same magnitude.” The system we advocate provides 16,384 bits (all data) per memory access, thus much more parallelism can be applied to many problems. [The Goodyear Aerospace Massively Parallel Processor (MPP), circa 1978, incorporated a memory bandwidth capability of 16,384 bits per access.]

The SIPP virtually eliminates the need for the following typical real time processing tasks, which are absolutely required in various multiprocessor configurations:

- Process or scheduling software
- Task assignment software
- Queue management software
- Data assignment software
- Memory coherency management software
- Table and data locking software
- Cache management software
- Multitasking software
- Indexing software
- Sorting software
- Data pointer assignment and update software
- Iterative processing of a set of data

The SIPP completely eliminates the processing time needed to execute the above software in a multiprocessor. For example, in a multiprocessor, flight plan update and conformance of the plan with the sensor developed track requires requires a loop instruction, executed once for each flight plan. Only a single execution of simpler software (no loops) is needed in an AP for all flight plans. For a 4,000 track environment, the MP requires 46,000 memory accesses for data. Only 506 accesses are needed in the SIPP architecture. The improvement in throughput, which is directly related to the number of memory accesses, is about 9,000% or 90 times greater [13]. When instruction access is considered, the improvement in throughput is more than 200 times greater.

The SIPP has an orders of magnitude higher data bandwidth than the MP because it's single instruction can access tens of thousands of operand bits simultaneously. The memory to processor bandwidth in an SIPP machine to handle the postulated environment is greater than 50 gigabytes per second[3]. And it is all data! This overcomes the limitations implied by Amdahl’s law.

Conclusion

We have demonstrated the Air Traffic Control scheduling problem is tractable; that is, it has a polynomial time solution. We have also explained why the current approach for implementing the solution to the ATC scheduling problem in a multiprocessor architecture results in an intractable problem. Additionally, we have shown that the simpler alternative approach of solving the Air Traffic Control problem using an associative processor results in a tractable solution and a feasible and statically predictable ATC scheduling algorithm.

In [12], Stankovic writes:

> “A highly *integrated and time-constrained resource allocation approach* is necessary to adequately address timing constraints, predictability, adaptability, correctness, safety and fault tolerance. For a task to meet its deadline, resources must be available in time and events must be ordered to meet precedence constraints. Many coordinated actions are necessary for this type of processing to be accomplished on time. The state of the art lacks completely effective solutions to this problem.”

This paper clearly demonstrates a new paradigm that provides a complete and effective solution to the ATC scheduling problem. Additionally, this same paradigm can currently satisfy the requirements of many real-time Command and Control System problems. We can see no alternative solution to the fully predictable, highly reliable, AP architecture for today’s command and control scheduling problems.

[3] 40 nsec access time.

References

1. C. J. Date, "An Introduction to Database Systems", Seventh Edition, Addison Wesley Publishing.
2. http//www.clbooks.com/ May 1995
3. http://home.columbus.rr.com/lusch/rtudslide02.html, Tom Lusch's site discusses a mid-air collision.
4. C.Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", J. ACM Vol 20, No 1 Jan 1973 pp. 46-51.
5. John Stankovic, Marco Spuri, Krithi Ramamritham, Giorgio Buttazzo, "Deadline Scheduling for Real-time Systems", Kluwer, 1998.
6. J. A. Stankovic, M. Spuri, M. Di Natale, G. C. Buttazzo, "Implications of Classical Scheduling Results for Read-Time Systems"|, Computer June, 1995, pp. 16-25.
7. M.H. Klein, J.P. Lehoczky, and R. Rakumar, "Rate-Monotonic Analysis for Real-Time Industrial Computing", IEEE Computer, pp. 24-33, 1994.
8. J.Potter, W. C. Meilander "Interarchitecture Comparative Analysis", Conference CIC'2000, Paper 274CCS.
9. M. R. Garey, R. L. Graham and D. S. Johnson "Performance Guarantees for Scheduling Algorithms", Operations Research, Vol 26, No.1 Jan-Feb, 1978.
10. W. C. Meilander, J. W. Baker, "ATC Computers - Yesterday, Today, Tomorrow", The 43rd Annual Air Traffic Control Association Fall conference Proceedings, 1998, pp. 91-95.
11. J. L. Potter, "Associative Computing - A Programming Paradigm for Massively Parallel Computers," Plenum Publishing, New York, 1992.
12. J. A. Stankovic, "Real-Time and Embedded Systems," The Computer Science and Engineering Handbook, Ed. Allen B. Tucker, Jr., CRC Press, 1997, pp. 1709-1724.
13. Will C. Meilander, Jerry Potter, Kathy Lizcka, Johnnie Baker " Real-Time Scheduling in Command and Control", MWPP workshop, Aug., 1999, http://www.mcs.kent.edu/~parallel.
14. Gregory F. Pfister in "In Search of Clusters - The Coming Battle in Lowly Parallel Computing", Prentice Hall, 1998, pg 516.
15. Editorial April 19, 1999, USA Today.
16. Lu, Qian. "Complexity Analysis of ATC." Thesis for Master's degree, Kent State University, Dec. 1997, sponsored by W. C. Meilander and Johnnie Baker.
17. J. L. Potter and W. C. Meilander, "Array Processor Supercomputers," IEEE Spectrum, vol.. 77, no. 12, pp. 1897- 1914, Dec. 1989.
18. Amdahl, G., "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," Spring Joint Computer Conference, 1967, pp. 483-485.
19. Martin Torngren, "Fundamentals of Implementing Real-Time Control Applications in Distributed Computing", Real-Time Systems, 14, pp. 219-250, 1998, Kluwer Publisher.
20. M. L. Dertouzos, "Control Robotics: the Procedural Control of Physical Processes" Information Processing 74, North Holland Publishing.
21. Ching–Chih Han and Kwei–Jay Lin, "Scheduling Parallelizable Jobs on Multiprocessors", Symposium on Real-Time Systems, 1989.
22. J.L. Potter, J.W. Baker, S. Scott, A. Bansal, C. Leanguksun, and C. Asthagiri, "ASC: An Associative Computing Paradigm, Computer", 27(11), 1994, 19-25.

23 J. A. Rudolph, "STARAN A Production Implementation of an Associative Array Processor", Fall Joint Computer Conference, December 1972.

24 J. R. Jackson, "Scheduling a Production line to Minimize Maximum Tardiness" Research Report 43, Management Science Research Project, University of California, Los Angeles, 1955.

25 CD covering Combined IEEE Real-Time Systems Symposia 1979-1999.