Proceedings of the 1995 Winter Simulation Conference
ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman

A DISTRIBUTED SIMULATION MODEL OF AIR TRAFFIC IN THE NATIONAL AIRSPACE SYSTEM

Eric L. Blair
Frederick P. Wieland
Anthony E. Zukas

The MITRE Corporation
Center for Advanced Aviation System Development
7525 Colshire Drive
McLean, Virginia 22102-3481, U.S.A.

ABSTRACT

The study of air transportation systems is a prime opportunity for the application of simulation modeling. The complexity and highly interdependent nature of the national air transportation system makes it difficult to study, even through simulation. As a result, models of this system are typically large and require long run times. This paper reports on a model designed to apply the emerging technology of parallel/distributed computing to facilitate the rapid analysis of national-scope air transportation systems. The model makes extensive use of the SPEEDES distributed simulation support software developed at the Jet Propulsion Laboratory by Dr. Jeff Steinman.

1 INTRODUCTION

An important measure of performance for air travel is delay. Delays costs the travelers time and the airlines money. This is a critical issue for the transportation industry in today's highly competitive (and mostly unprofitable) business environment. Although the "friendly sky" may appear to be unbounded, the US air traffic system has a finite capacity with significant choke points at major airports. Since the commercial airlines use the same aircraft to make a sequence of flights, a delay on one flight leg will propagate a delay on the connecting flights. Most experienced air travelers have found ample opportunity to practice their language skills while waiting for the arrival of the flight being executed by the aircraft assigned to their flight.

Because the air transportation system is large and complex, simulation models developed to study it are large and typically require powerful computers and long run times. This paper reports on a model development project that has successfully employed the emerging technology of parallel/distributed simulation to the design and implementation of a national-scope air traffic simulation model.

Section 2 describes the system that is the subject of this modeling effort, the National Airspace System. Related work in this domain of application is reviewed in Section 3. This model relies heavily upon object-oriented design and the SPEEDES (Steinman 1994) software supporting parallel/distributed simulation. These aspects of the model are presented in Sections 4 and 5. Section 6 continues the discussion of our model and its distribution scheme. Section 7 gives preliminary performance results for the model with respect to execution times as a function of the number of processors on which the model is run. Finally, Section 8 suggests some appropriate conclusions from our work.

2 THE NATIONAL AIRSPACE SYSTEM

The National Airspace System (NAS) can be described as the collection of airports and airfields, navigation fixes, sectors, and air routes that facilitate air travel across the continental United States. The vast majority of air traffic, both commercial and general aviation (GA), follows paths in the network of routes defined by airports and navigation beacons. Flight paths intersect at fixes that are defined by ground based navigation aides. All flights in the continental US are conducted under the control of the Federal Aviation Administration (FAA) Air Traffic Control (ATC) services. All commercial and most business aircraft are required to fly under instrument flight rules (IFR). The FAA provides a number of services for IFR flights including the approval of flight plans and the issuance of instructions that assure proper separation between aircraft will be maintained. These control instructions involve the assignment of route, flight level, speed, and admittance to zones of airspace referred to as sectors.

The network of air routes is defined by paths projecting straight lines on the earth's surface and connecting navigation fixes. A fix is the projection of

an imaginary point up into space over a land-based navigation beacon. Two types of fixes deserve special attention as they serve to organize the flow of aircraft into, and out of, an airport. The "arrival fix" collects incoming aircraft as they approach the airports for a landing. The "departure fix" is a point from which departing aircraft will span-out to join connecting air routes. The purpose of arrival and departure fixes is to ensure proper spacing between aircraft on the path segment between the fix and the airport.

Sectors are irregularly defined three-dimensional volumes of space that are constructed to distribute the ATC workload so that a sector can be managed by a single controller (or controller team). Sectors can be stacked on top of each other. All air routes pass through contiguous sectors. A sector that covers an airport is called a "terminal air space." Sectors that cover the airspace between and around terminal sectors are referred to as "en route sectors" and are controlled by one of 20 en route ATC centers. Sectors have capacities defined as the number of aircraft that can be simultaneously resident in the sector; this number is determined by the size of the sector and complexity of the route structure within it. To enter the next sector, a pilot must receive permission from the controller working the requested sector. Transitioning from one sector to another is referred to as a "hand-off" since it is coordinated among the pilot and the controllers of the departed and entered sectors.

From the above description, it is clear that if we ignore the second-order congestion effects induced by separation requirements within a sector, the NAS can be modeled as a network of queues. Each airport, sector, and fix is a queue node. Links between nodes are defined by the air routes that pass sequentially through sectors en route from one airport to another. Airports serve as source nodes for originating flights, service nodes for continuing flights, and sink nodes for terminating flights.

The scope of the NAS is of some interest. There are approximately 16,000 airfields in the continental US, where an "airfield" is defined as any place where an airplane can take off and land. Airfields run the gamut from crop-duster sites to major airports like Chicago O'Hare. Of the 16,000 airfields, approximately 1,000 handle all of the commercial carrier air traffic; the top 500 most active of these handle 80% of all the GA traffic. Of the top 500 GA airfields, there are only about 60 airports where resource contention cause major arrival or departure delays for either commercial or GA flights. Additionally, there are over 700 airspace sectors in the NAS. A typical day's air traffic consists of 35,000 flights by the airlines and from 10,000 to 16,000 GA and military flights.

3 RELATED WORK

A number of different models have been constructed to study the NAS or subsections of it; they differ with respect to analysis objectives, level of detail and embedded assumptions. The Reorganized ATC Mathematical Simulator (RAMS) model (Czech and Crook 1994) is primarily geared toward analysis of air traffic control operations and training of air traffic controllers. The principal performance measure derived from the RAMS model is the level of air traffic controller workload, defined as the percentage of time, over an hour interval, that a controller is busy handling ATC operations. The RAMS model contains very detailed information about the geometry of airspace sectors, the path of aircraft through the airspace, and the approach paths to airports. The RAMS model could, for example, be used to determine the capacity of individual airspace sectors. RAMS was developed for Eurocontrol applications.

SIMMOD (Hargrove-Gray 1994) is another example of a complex air traffic simulation model. It was developed by CACI and American Airlines under the sponsorship of the FAA. SIMMOD is a very appropriate tool with which to study a single airport or set of closely located airports (e.g., the New York City area). It contains a detailed model of the physical layout of each airport and surrounding airspaces included in the study. Airport and airspace description are all provided through data files (CACI 1991). The volume of data required to model an airport such as Dallas-Fort Worth's DFW is quite large. SIMMOD has been used very effectively by American Airlines (among others) to model operations at hub airports. Due to the large data requirements and the detail of activity modeled, SIMMOD is less effective for the analysis of NAS-scope studies.

The National Airspace System Performance Analysis Capability (NASPAC) has been under development by the MITRE Corporation since 1987 (Frolow and Sinnott 1989, Millner 1993). NASPAC is a low-resolution model of the entire NAS. Although all flights are represented in a NASPAC run, only 58 airports are modeled in detail; these are the top 50 airports with respect to volume of IFR operations plus 8 smaller airports with airspace overlapping with one or more of the top 50. The primary purpose of NASPAC is to measure the delays and to capture the "delay ripple" caused when aircraft that are delayed cause further delays "downstream" on the flights in their itinerary. Supporting software modules are used to create air routes and itineraries (lists of connecting flights) for each simulated aircraft off-line from the simulation.

The simulation models cited above are all designed and programmed to be executed sequentially on a single processor. To our knowledge, there is no model of air traffic, other than the work presented in this paper, designed to be run as a parallel/distributed simulation.

4 MODELING THE NAS WITH OBJECTS

Computer simulation has historically been an area in which object-oriented technology has been used effectively to exploit the analogy between objects in the real word and model components implemented as software objects. The simulation programming language SIMULA (Birtwistle and Palme 1974) is often cited as the first object-oriented language. The elements of the NAS are well suited to this approach. Our model consists of a number of objects programmed in C++. These objects can be classified into two distinct categories that correspond to the traditional concepts of "permanent" and "temporary" entities used by most simulation software. Here the temporary objects are associated with the aircraft and the information necessary to determine its path through the NAS network. Permanent objects correspond to elements of the network itself (airports, sectors, and fixes), and collections of aircraft (queues). Each NAS object is composed of data (attributes) and methods (functions that access or change the data). Simulation events are associated with the NAS objects and, due to the conventions of SPEEDES (reference Section 5), are also objects themselves. The major objects and events are described below. A number of supporting objects from which these objects are derived (i.e., base class objects) are not discussed (reference Blair et al. 1994 for details). It should also be noted that to a large degree, the design of our NAS model objects was strongly influenced by the operational concepts of the SPEEDES software that implements parallel/distributed simulation.

4.1 TEMPORARY OBJECTS

The NAS simulation has three objects for which instances are created and destroyed many times within the simulation run. These objects are all associated with management of an aircraft as it executes flights in its itinerary.

4.1.1 The *aircraft* Object

The primary temporary object is the *aircraft*. The *aircraft* object is the basic token used to represent traffic as it flows through the NAS network. Each *aircraft* object contains information pertaining to the aircraft's identity, state, flight plan, and reference time values. Identification includes the associated airline company (if any) and the equipment type (e.g., Boeing 737, 777, etc.). Aircraft state descriptors are speed and status; the status attribute is an integer used to identify different activity states such as "in-flight," "at-gate," "holding," etc. Each *aircraft* has an itinerary attribute (an object) that lists the sequence of flights that are to be flown by this aircraft. A flight attribute (also an object) defines the flight path as described below. Reference time values are used to calculate utilization and delay times associated with aircraft, airports, sectors and fixes.

4.1.2 The *flight* Object

The *flight* object is analogous to a flight plan in the real world. The *flight* object indicates the sequence of fixes and sectors to be visited between origin and destination airports and determines an aircraft's routing through the NAS queueing network. It consists of an ordered set of records. Each record is a quad-tuple composed of a reference to a NAS permanent object (airport, fix, or sector), an integer key identifying the NAS object type, and two real valued numbers determining the occupancy time for the NAS object and the time to the next NAS object.

4.1.3 The *itinerary* Object

The *itinerary* object is a list of sequential flights that are to be executed by a single aircraft. It contains an ordered set of records, with each record defining a flight by the origin and destination airports. Flights in the model (and the real-world NAS) can be divided into two groups: commercial and GA. Commercial flights are flown by the airline carriers (e.g., American, Delta, Southwest, etc.) and are listed in the Official Airlines Guide (OAG). GA flights are made by private pilots typically in smaller private aircraft. Military flights are grouped with GA traffic. The concept of the *itinerary* object was introduced to accommodate a more accurate modeling of airlines operations in which flights are scheduled so that they can be flown by the same aircraft. This captures the ripple effect caused by a delayed arrival inducing a delayed departure of the next flight for a commercial aircraft. GA flights do not typically dove-tail flights and are modeled as independent activities. GA *aircraft* object will have an *itinerary* that references a single flight.

Four time value attributes are included in the *itinerary* object to be used to compute flight delays. These attributes store scheduled and actual departure and arrival times. Scheduled times are taken from the OAG for airline flights. For GA flights, the scheduled departure time is generated randomly (as described below) and the scheduled arrival time is computed by adding the estimated undelayed travel time.

4.1.4 Generating Itineraries and Flights

Commercial flights from the OAG are grouped into itineraries by an off-line program originally developed for the NASPAC simulation model. This program attempts to match flights for the same carrier and equipment class

that could logically be scheduled in sequence. That is, the scheduled arrival of the first flight must precede the scheduled departure of the second by a minimal period of time at the same airport. The period of time between flights is referred to as the "turn-around time" and varies from airline to airline and airport to airport. Itineraries are sorted by the airport of origin for the first flight; one file is constructed for each modeled airport and itineraries are listed in increasing order of first flight scheduled departure time. The initial scheduled departure of each commercial aircraft is introduced into the simulation as an "external" event.

GA flights (an itinerary of a single flight) are generated during the simulation at each airport. The process that generates these flights is a non-homogeneous Poisson process for which the rate parameter varies with each simulated hour. The hourly rates were determined from an analysis of 220 days of actual traffic data and are specific for each airport.

The *flight* object is constructed before an aircraft is scheduled to takeoff from the origin airport. The sequence of NAS objects (fixes and sectors) that the aircraft will visit en route to the destination airport and the transit times for each are read from a table that is loaded into memory at simulation start-up.

4.2 PERMANENT OBJECTS

The permanent objects of our simulation model all have geometric and geographical significance. They represent the service/queue elements of the NAS queueing network.

4.2.1 The *airport* Object

The *airport* object provides data and behavior necessary to initiate and terminate flights. Commercial flights are introduced through external events as described above. Each such external event is invoked by an itinerary read from an airport-specific file. Itinerary and aircraft objects are created; the *itinerary* is attached to the *aircraft* and the departure for the first flight is scheduled. Aircraft that pass through an airport as part of their itinerary are stored in a list, which is part of the *airport* object, where they wait until the scheduled departure time or until a minimum turn-around time has expired (for those aircraft that are behind schedule).

Each *airport* object maintains two queues and two service processes: one set for arrivals and one for departures. Arrival and departure operations can be carried out simultaneously. Arrival and departure rates are determined jointly and dynamically using an algorithm that reflects the real-world capacity allocation process used to balance the arrival and departure queues. This algorithm was taken from the NASPAC model and is documented in (Frolow and Sinnott 1989).

Although all airports share the same base object class, there are three derived types. Each is used to permit a different level of detail in describing airport operations and measuring performance. The "detailed" airport has well defined capacities, limited queue space, and performance statistics are collected for utilization and delays. The top 58 busiest airports are modeled as detailed *airport* objects. The "generalized" airport has infinite capacities and infinite queue size; no queueing develops and no facility related delays are recorded. The top 500 busiest airports in terms of GA traffic (excluding the 58 detailed airports) are modeled as generalized *airport* objects. All remaining commercial traffic directed from/to airports not included in the above 500 are handled by a set of 20 "source-sink" *airport* objects. Source-sink airports are similar to generalized airports except each handle traffic from/to multiple airports. This is particularly useful for introducing and terminating international flights.

4.2.2 The *sector* Object

The *sector* object is a multi-server queueing system. It has a finite capacity intended to reflect workload constraints for air traffic controllers. When a sector is full, aircraft seeking to enter the sector are blocked and must remain in the NAS object that they are currently occupying. The time to cross a sector (i.e., "service time") is included in the information contained in the *aircraft* object's *flight* object. The *sector* object has attributes used to describe its state and measure its utilization. There are over 700 en route sectors in the NAS model.

4.2.3 The *fix* Object

The NAS model's *fix* object provides the function of separation of aircraft on arrival to, and departure from, an airport. By spacing aircraft, they also determine the arrival and departure rates. In the real world, arrival and departure fixes are used to match these rates with airport capacity and to insure minimal safe in-trail separation between aircraft. The *fix* object is a single server queue. The service process in the model corresponds to the act in the real NAS of an aircraft passing over the fix and progressing far enough along the aircraft's path to maintain a specified miles-in-trail separation restriction for traffic over the fix. Arrival *fix* objects enforce miles-in-trail restrictions for aircraft converging on an arrival approach. Departure *fix* objects enforce miles-in-trail restrictions for aircraft taking off and approaching a span-out point. The service time is determined by the miles-in-trail restriction and the *aircraft* object's speed.

4.3 Model Mechanics

The conceptual design underlying the NAS simulation model is that of a queueing network. Queue nodes in the network are the airport and sector objects. An airport operates as a pair of G/G/1 queues, one for arriving and

one for departing traffic. Airspace sectors function as single G/G/n queues. This simple, high level conceptual model, that of a network of queues describing the various elements of the NAS, is remarkably elegant for capturing the necessary delays and the delay ripple effect of the buildup of delay for commercial aircraft. There are, of course, other details in the model, including a truncated Gaussian distribution used to model the uncertainty in gate push-back times, parameters for describing taxi-in and taxi-out times at the various airports, provisions to model "ground-hold" programs instituted for flow-control purposes, and means with which to adjust airport and sector capacities for weather patterns that develop during the day.

5 SPEEDES

Synchronous Parallel Environment for Emulation and Discrete-Event Simulation (SPEEDES) is software that enables a parallel simulation to be executed in distributed mode on a network of UNIX workstations. SPEEDES was developed at the Jet Propulsion Laboratory by Dr. Jeff Steinman and is reported in (Steinman 1991, 1992, 1993, 1994). A port of SPEEDES to the Intel Paragon parallel computer has been recently completed.

SPEEDES provides a ***logically correct*** parallel simulation environment, meaning that the results of running a simulation in parallel on multiple processors is the same as that produced by running the same simulation as a serial program on a single processor. A number of algorithms (i.e., synchronization protocols) have been developed to produce logically correct simulations. SPEEDES implements several synchronization algorithms from the class of algorithms described as *optimistic*.

The unit of control in a parallel simulation is the logical process (LP). Each LP executes a set of events (associated with some physical process of the system being simulated). Logical processes are software entities and are distributed among the computer processors of the network (or parallel computer) for execution. Events can be created by one LP for execution by another LP. In this case, the arguments to the event routine are packed into a message and sent to the target processor. Under this scheme, it is possible for the individual LP clocks to get out of synchronization; when this happens, an event message can arrive at an LP scheduled for execution in the LP's past. Optimistic protocols solve this problem by undoing events executed out of sequence and starting over with the clock reset to the time of the precipitating event. Since erroneously executed events may have scheduled other events, "undoing" may require the chasing down and correction of many cascading events. This process of undoing events is called "rollback." There are several strategies that have been developed to reduce the burden of rollback including Time Warp (Jefferson 1985), Moving Time Windows (Sokal et al. 1988), Breathing Time Buckets (Steinman 1991), Breathing Time Warp (Steinman 1993) and the Synchronous Parallel Simulator (Peterson and Chamberlain 1993). SPEEDES supports all of these except Moving Time Windows. The SPEEDES software is described in detail in (Steinman 1994).

6 DISTRIBUTING THE SIMULATION

The parallel computational model provided by SPEEDES differs sharply from that underlying most other distributed simulation protocols; this difference we view as a strength. The standard view is that the physical model is decomposed into a *set* of logical processes (LPs), each of which contains methods that describe the events acting upon the LPs (see, for example, Fujimoto 1990). This computational model is often implemented using the object-oriented paradigm. SPEEDES presents a radical departure from this standard model. With SPEEDES, the LPs and the events that act upon them are separated into different objects. The object that is conventionally associated with the LP is called the "data object," because it contains the state data that must be saved and restored during processing and rollback. The data object contains methods to access, modify, and compute simple functions on the data. The object that is created to facilitate the event is called, appropriately enough, the "event object." This object contains the main logic for the event, including the ability to access its associated data object, change its state, schedule other events, and write output to files or output devices.

In the SPEEDES computational model, the data objects are unaware that they are part of a simulation. They do not contain the notion of simulation time, and they do not schedule events for other objects. This separation of the "data" from the "events" is actually a strength. The data objects could easily be part of a database system as well as a simulation system; because simulation requires a enormous amount of data, the same objects can be reused for both purposes without recoding or reimplementing. New events can be defined without recompiling the data objects (provided that the new events do not add new data requirements to the data objects).

The function and coordination of the data and event objects are described by a SPEEDES diagram (Figure 1). The operation of the arrival and departure *fix* object has been omitted to simplify the exposition. The *fix* object can be modeled as a *sector* object with a capacity of one.

The SPEEDES diagram shows the two data objects (one for airports, one for sectors) as rectangular boxes. The event objects are shown as the octagonal boxes. The association between the events and the data objects are shown by a thin line connecting the two; if an event schedules another event, there is an arrow whose tail is the scheduling event and head is the scheduled event. The arrow is labeled with the lookahead value of the event; lookahead is the minimum time difference between the simulation time at which the message is created ("send

time") and the simulation time at which the event is to be executed ("receive time").

The events surrounding the *airport* data object are responsible for departures and arrivals at the airport, as well as the generation of GA flights. The events surrounding the *sector* data object are all associated with the sector-to-sector hand-off protocol. If an aircraft is not delayed, then a series of ***handle requests*** event messages pass the aircraft from one sector to another. When a delay is encountered, a ***handle reject*** event is scheduled for the "exited" sector and an ***inspect queues*** event message is sent to schedule this event for the "entered" sector. When the "entered" sector is finally clear, a ***handle accept*** event message is sent to the "exited" sector to schedule the immediate release of the aircraft.

The arrows are coded to show which NAS objects are affected by each event. Events may be scheduled by one NAS object for the next object to be visited by the aircraft; these events are indicated by the solid line indicating the flow of an associated event message. Sometimes an event message flows from a next-to-be-visited NAS object to a currently visited object. This is the case when an aircraft is denied entry into the next sector; the arrow associated with this message is shown as a heavy dark line. When an event is scheduled by the current NAS object for itself, the arrow is shown as a light gray line.

There are three factors that are critical in determining the success of a parallel/distributed simulation model. They are event granularity, degree of LP interdependency, and workload balancing. Granularity refers to the relative time to process an event as compared to the time to pass event messages between processors. Granularity is largely a property of the system being modeled and is not easily altered by the implementation without significantly impacting the model detail and fidelity. Independence among LPs can, however, be greatly affected by model design. The use of a "geographic region" concept in our model is used to organize load balancing. The transfer of aircraft among NA S objects belonging to the same region is accomplished without constructing and sending a message from one physical processor to another. External messages (from one processor to another) only need to be sent when an aircraft leaves a NAS object located in one region to enter a NAS object located on another processor.

For the NAS model, the airspaces associated with the 20 Air Route Traffic Control Centers (ARTCCs) provide a natural scheme with which to regionalize. If each LP corresponds to an ARTCC, then the simulation can be distributed among up to 20 physical processors. With less than 20 physical processors, load balancing can be achieved by grouping ARTCC regions for assignment based on an even distribution of the estimated traffic levels.

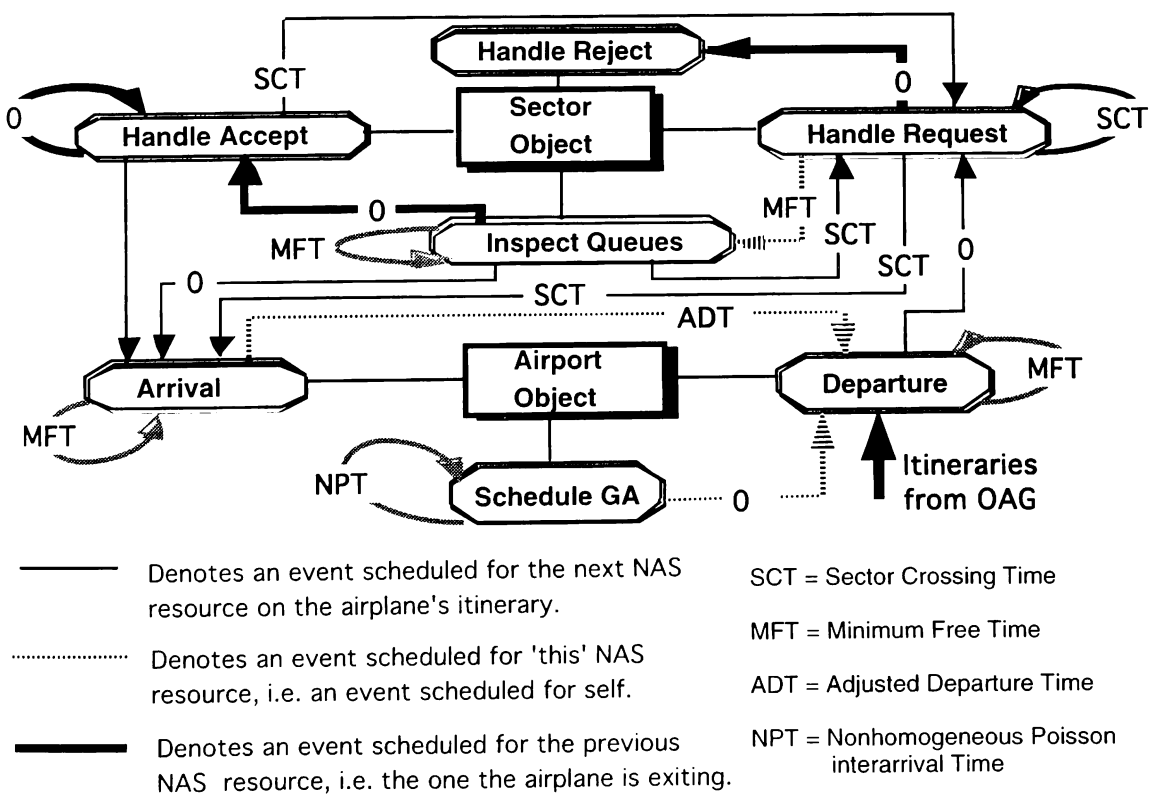

Figure 1. Software Architecture of the System, Shown as a SPEEDES Diagram.

7 PERFORMANCE RESULTS

The objective of a parallel/distributed simulation model is to reduce run times by distributing the computation over a number of processors; the more processors, the smaller the run times. A very successful parallel implementation will show run times that are inversely proportional to the number of processors, e.g., going from one to two processors results in run times that are half as long. This behavior is referred to as "scalar" or "linear." However, as more processors are added, the ratio of time spent executing events to that spent processing messages must decrease and performance will degrade, resulting in sub-scalar behavior. Figure 2 displays a graph of the run times for the simulation of a single NAS day using from one to nine processors. These results are preliminary and lack precision. However, they clearly demonstrate a decrease in run time with an increase in the number processors used. The excessively high results for four and five processors is believed to be due to poor load balancing. In each case the same scenario was used with the same random number seeds. The scenario included over 50,000 flights. Employing nine processors resulted in better than 50% reduction in run time compared with a serial execution on a single processor. A much more complete review of performance issues, including discussion of all five protocols available with SPEEDES, is presented in (Wieland et al. 1995).

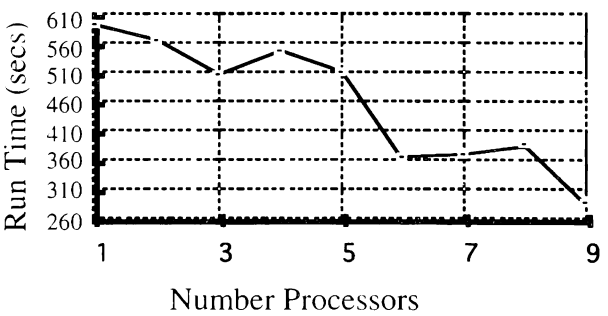

Figure 2. Simulation Run Time as a Function of the Number of Processor.

8 CONCLUSIONS

We found the use of parallel/distributed simulation methods to be highly productive for our application to air traffic modeling. Preliminary results predict that we will be able to run a NAS day simulation in less than 3 minutes on a network of less than 20 SUN workstations. This has three implications for future work. First, a small run time per replication makes it practical to conduct experiments with many replications; this allows greater exploration of alternatives in performance studies, better precision of simulation results and assessment of system and sample variability. Secondly, by making the run times smaller, we significantly reduce our response time with respect to undertaking new studies; this makes simulation a much more versatile tool for systems analysis. Thirdly, the present model, although quite large, is course grained with respect to the detail of representation of air traffic control and traffic flow management operations. Increasing the detail will increase the computation. The present distributed simulation model provides a base from which to build more complex models. There is adequate room for increasing the level of modeled scope and detail while still satisfying practicality constraints on run time.

A few comments regarding model structure are in order. First, it is SPEEDES-specific; a design for another parallel/distributed protocol would probably look very different. Secondly, the SPEEDES induced model structure is substantially more complex than a sequential model design such as that of NASPAC. We feel that the performance gain is worth the pain. SPEEDES is a remarkably fast and versatile simulation system. We have found it to be robust in the presence of dynamic memory operations, both in the formulation of messages and in constructing a data object's state. The incorporation of multiple synchronization protocols within SPEEDES allows for easy exploration and selection of the best protocol for a specific model and study scenario; the options available exhibit a wide range of performance characteristics. Although the SPEEDES computational model differs from the more conventional approach proposed by the parallel/distributed simulation community, the differences allow for more flexibility in the design and development of a simulation.

Finally, we observed that performance is greatly increased by introducing look-ahead in the scheduling of events. This is not a new conclusion, but we found it to be very relevant to our model. The most obvious way to affect a sector-to-sector hand-off would be to send a "request" message to the next *sector* object requesting access; this message would be followed by a "response" message from the next object indicating acceptance or rejection. The elapsed simulation time between these messages is zero. This query process induces a strong dependency between the associated LPs seriously degrading concurrent processing. Because such hand-offs are a dominant part of the NAS simulation, this is an important design consideration. Thus, our events were constructed to anticipate future sector hand-offs and to assume that the hand-offs were successful, unless told otherwise by a ***handle reject*** message. The presence of these extra events, which are due to the general paradigm of parallel/distributed simulation, adds code to the model implementation, requires more time for verification, and increases the burden for maintenance and documentation. This extra effort is worth the price only if the performance increase is large.

ACKNOWLEDGMENTS

The authors would like to acknowledge the contributions of Jeff Steinman of JPL who was instrumental in providing access to the SPEEDES software and assisted in the crafting of our application to fit the SPEEDES computation model. We would also like to thank Ron Haggarty, David J. Chadwick, Myra Jean Prelle and Edward H. Bensley of the MITRE Corporation for their encouragement, insight, and general guidance.

REFERENCES

Birtwistle, G. and J. Palme. 1974. *SIMULA Language Handbook*. Swedish National Defense Institute, Stockholm, Sweden.

Blair, E. L., F. P. Wieland, and A. E. Zukas. 1994. The Detailed Policy Assessment Tool (DPAT): A Parallel Simulation of Capacities and Queueing Delays in the National Airspace System. MITRE Technical Report MTR 94W200, McLean, VA.

CACI Products Company. 1991. SIMMOD release 1.2 User's Manual.

Czech, H. C., and I. Crook. 1994. RAMS: A Flexible Modeling Tool for the Simulation of ATC Environments in Europe via the Application of Object-Oriented Technology, In *Simulation Conference 20,* CACI, Inc. Washington, DC.

Frolow, I. and J. H. Sinnott. 1989. National Airspace System Demand and Capacity Modeling. In *Proceedings of the IEEE*, 77:1618-1624.

Fujimoto, R. M. 1990. Parallel Discrete-Event Simulation. *Communications of the ACM* 33:30-53.

Hargrove-Gray, B. 1994. SIMMOD. In *Simulation Conference 20.* CACI, Inc. Washington, DC.

Jefferson, D. 1985. Virtual Time. *ACM Transactions on Programming Languages and Systems* 7:404-425.

Millner, D. C. 1993. Design of the NASPAC Simulation Modeling System. MITRE Technical Report MTR 92W135, McLean, VA.

Peterson, G. D. and R. D. Chamberlain. 1993. Exploiting Lookahead in Synchronous Parallel Simulation. In *Proceedings of the 1993 Winter Simulation Conference*, ed. G W. Evans, et al., 706-712.

Sokal, L. M., D. P. Briscoe, and A. P. Wieland. 1988. MTW: A Strategy for Scheduling Discrete Simulation Events for Concurrent Execution. In *Proceedings of the SCS Multiconference on Distributed Simulation*, ed. B. Unger and D. Jefferson, 19:34-42.

Steinman, J. S. 1991. SPEEDES: Synchronous Parallel Environment for Emulation and Discrete Event Simulation. In *Advances in Parallel and Distributed Simulation, SCS Western Multiconference*, ed. V. Madisetti, D. Nicol, and R. Fujimoto, 95-103.

Steinman, J. S. 1992. SPEEDES: A Multiple-Synchronization Environment for Parallel Discrete-Event Simulation. *International Journal in Computer Simulation* 2:251-286.

Steinman, J. S. 1993. Breathing Time Warp. In *Proceedings of the 1993 Workshop on Parallel and Distributed Simulation (PADS93)*, ed. R. Bagrodia and D. Jefferson, 109-118.

Steinman, J. S. 1994. *SPEEDES User's Guide, Beta Version 2.0.* The MITRE Corporation and the Jet Propulsion Laboratory.

Wieland, F. P., E. L. Blair, and A. E. Zukas. 1995. Parallel Discrete-Event Simulation (PDES): A Case Study in Design, Development, and Performance Using SPEEDES. In *Proceedings of the 1995 Workshop on Parallel and Distributed Simulation (PADS95).* 103-110.

AUTHOR BIOGRAPHIES

ERIC L. BLAIR is a Lead Engineer in the Modeling and Analysis Technology Department of MITRE's Center for Advanced Aviation System Development (CAASD). Prior to joining MITRE, he held faculty positions at North Carolina State University, Rensselaer Polytechnic Institute, and Texas Tech University.

FREDERICK P. WIELAND is a Senior Scientist in the Modeling and Analysis Technology Department of CAASD. He has been closely associated with the field of parallel/distributed simulation since his participation as a member of the Time Warp development team at the Jet Propulsion Laboratory.

ANTHONY E. ZUKAS is a Lead Engineer and Group Leader in the Modeling and Analysis Technology Department of CAASD. Tony has a broad background in systems analysis and model design. Prior to joining MITRE, he participated in software development at UNISYS and Scientific Applications International Corporation.