Cluster Node Computing for Target Generation Systems in Aircraft Simulations

Spencer Monheim[1], Michael Feher[2], and James R. Murphy[3],
NASA Ames Research Center, Moffett Field, CA, 94035, USA

Bonnie Andro-Avila[4]
MIRACORP, Inc., Moffett Field, CA 94035

Target generation systems provide virtual position and state reports of aircraft for air traffic simulations. As the scope and scale of the simulation domains expand, there is a need to develop systems that can generate position reports for thousands of simulated aircraft simultaneously at high update rates that support out-the-window visualization. This paper discusses the motivation and reasoning behind investigating development of a next generation target generator through distributed computing using clustered node processing and the benefits such a target generation system has on future research that utilizes human-in-the-loop simulations.

I. Nomenclature

ATG	=	Airspace Target Generator
MACS	=	Multi-Aircraft Control System
ATM	=	Air Traffic Management
FFC	=	FutureFlight Central
Hz	=	Hertz (cycles per second)
UAM	=	Urban Air Mobility
SimLabs	=	NASA Ames Research Center's Simulation Laboratories
Go	=	The Go Programming Language
TCP	=	Transmission Control Protocol
IPC	=	Inter Process Communication
DASE	=	Distributed Aircraft Simulation Engine

II. Introduction

NASA utilizes air traffic management (ATM) simulations to evaluate potential future air transportation concepts and decision support tools intended to improve the efficiency and safety of the National Airspace System (NAS) and future, non-traditional airspace operations. Target generation systems are used to provide air vehicle position data for ATM simulations. Designed to compute the vehicle dynamics and provide positions of many aircraft simultaneously, they generally produce lower update rates than the flight simulators used for pilot training, which are focused on modeling a single vehicle's performance. High fidelity for a target generation system is desired for greater simulation realism and therefore, presumably, more accurate evaluation, and is represented in this study by the update rate. The fidelity for a target generation system is determined by the requirements for the given simulation and is always relative to the research questions under consideration. Presently NASA Ames Research Center's Simulation Laboratory (SimLabs) utilizes two different target generators, the Airspace Target Generator (ATG) [1], which can be used for airport surface and airspace operations, and the Multi-Aircraft Control System (MACS) [2] for airspace operations only. While both of these systems provide coverage for their intended domains, each has specific limitations which

[1] Aerospace Software Engineer, NASA Ames Research Center
[2] Aerospace Engineer, NASA Ames Research Center
[3] Project Manager, NASA Ames Research Center
[4] Technical Writer, NASA Ames Research Center

constrains their utility for simulations that span across multiple airspaces. In addition, both target generators were designed to be extremely effective for their use cases of driving radar simulations, but were not designed with the scale, number of aircraft a generator can handle, or update rates, for ensuring smooth out-the-window visuals, anticipated to be needed in future simulations. In particular, recent interest in Urban Air Mobility (UAM)[3] simulations is coupled by an increasing interest in smooth, out-the-window visualizations for human subjects and for realistic flight demonstration. SimLabs operates a simulation and visualization facility, FutureFlight Central (FFC), as seen in Figure 1, to simulate air traffic control operations from a control tower view-point.

Fig. 1 – NASA Ames Research Center's FutureFlight Central

UAM simulations are anticipated to require an order of magnitude greater traffic density than is currently typically modeled and simulated in FFC experiments. This requirement will necessitate that simulation facilities like FFC provide smooth out-the-window visuals, more closely to that typically found in cockpit flight simulators. Such simulators benefit from low-latency smooth out-the-window visuals due to not requiring extrapolation or interpolation to provide smoothing, or update at the rate of the displays, 30 Hz and 60 Hz being the most common.

In contrast, neither MACS nor ATG update rates (of 1 Hz and 4 Hz, respectively), is anticipated to meet the UAM simulation requirements of being able to provide smooth out-the-window visuals with the estimated number of aircraft. These target generators have performed adequately for most ATM simulations, which do not generally have hard real-time requirements, or the frame rate requirements that are common in flight simulators. Further, some ATM simulations may only require rates matching the update rates of real-life systems like Automatic Dependent Surveillance-Broadcast (ADS-B). Currently, both target generators burden the image generation systems at FFC with the need to heavily utilize extrapolation in order to best predict and approximate positional data between updates in order to smooth the out-the-window visuals.

This paper describes the design and development of a new target generation system that utilizes existing aircraft and trajectory models but provides a framework for managing a range of position update rates, while also increasing the number of aircraft that can be supported, through the inclusion of modern computing development practices and technology.

III. Background

For simulation of ATM tools and concepts, the Simulation Laboratory at NASA Ames Research Center [4] utilizes two different target generators, ATG and MACS, which were independently created by two groups at NASA Ames Research Center. ATG was developed in the 1980's/1990's from the Pseudo Aircraft System [5], to manage the traffic scenario for an ATM simulation, providing pilot stations used to support the "flying" of multiple virtual aircraft by a single pseudo pilot and connecting to radar displays for virtual controllers. It was used to conduct En Route and Terminal air traffic control simulations, providing position updates for aircraft once every 12 seconds, matching the Federal Aviation Administration's (FAA's) en route host computer system update rate. During later development, ATG increased the position report update rate to 4 Hz and added ground operations, as well as a ground pilot station, as seen in Figure 2, to enhance the initial tower out-the-window visualizations required for simulations run at FFC. MACS, like ATG, utilizes pilot stations, a radar display for controllers (as seen in Figure 3) and a simulation manager. MACS is used to simulate position reports from an ADS-B system and provides the data at a 1 Hz update rate. MACS is predominantly used for simulations of airspace operations. The controller station display and pseudo pilot display enable a simulated controller and pilot-in-the-loop air traffic simulation [6].

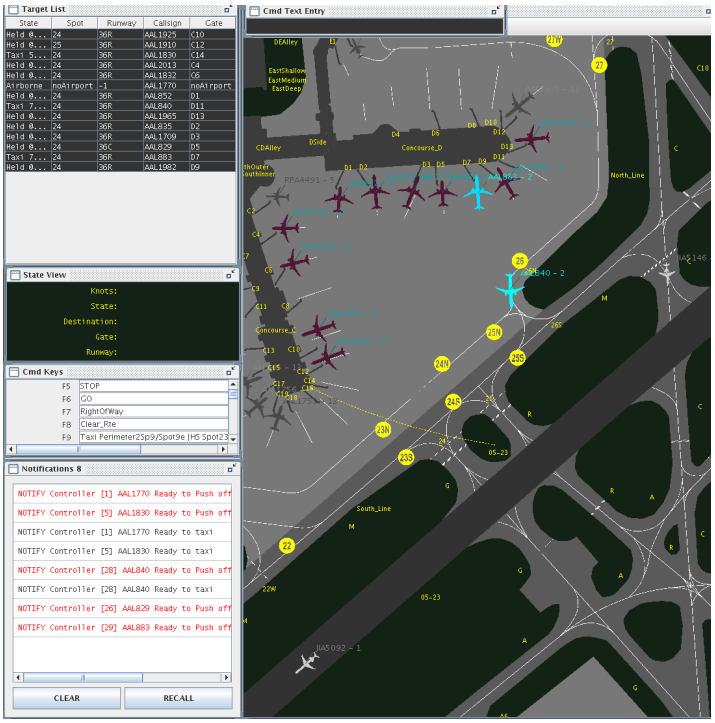

Fig. 2 - ATG Ground Pilot Station

Fig. 3 - MACS Controller Station

In addition to update rate concerns noted above, ATG and MACS have other limitations for future ATM simulations. MACS does not simulate airport surface operations. In addition, MACS development has been motivated by a wide range of applications, and so its capabilities have grown and diverged from one of its core functions as a target generator. The software baseline is highly fragmented and thus creates performance issues.

In comparison, ATG is capable of airport surface modeling. However, its development was also driven by specific projects, and its architecture is a 32-bit version that was not systematically modernized and thus limits scalability.

Both target generators were also developed with a focus on the pilot display to control aircraft. Though this capability is not required to generate targets, that is, aircraft position reports can be generated in an automated fashion by both tools, the underlying system architecture makes it cumbersome to scale the number of aircraft supported beyond a few hundred.

In order to leverage the best features of each system to accommodate the requirements of recent simulation experiments, SimLabs engineers have been utilizing MACS and ATG simultaneously, incorporating a complicated hand-off scheme. A hand-off area was defined and specifically adapted to the simulation airspace that continuously tests the location and timing of the hand-offs to ensure a smooth transition, as seen in Figure 4. ATG provides the aircraft position locations for aircraft within the three-dimensional polygon, while MACS provides targets outside the polygon. The handoff messaging and interactions between the target generators is facilitated by the High Level Architecture (HLA) [7] middleware. As an example, NASA's Airspace Technology Demonstration 2 (ATD-2) Human-in-the-loop simulations, MACS was used to handle the Center and Terminal airspace, while ATG handled the airport airspace and ground operations [8].

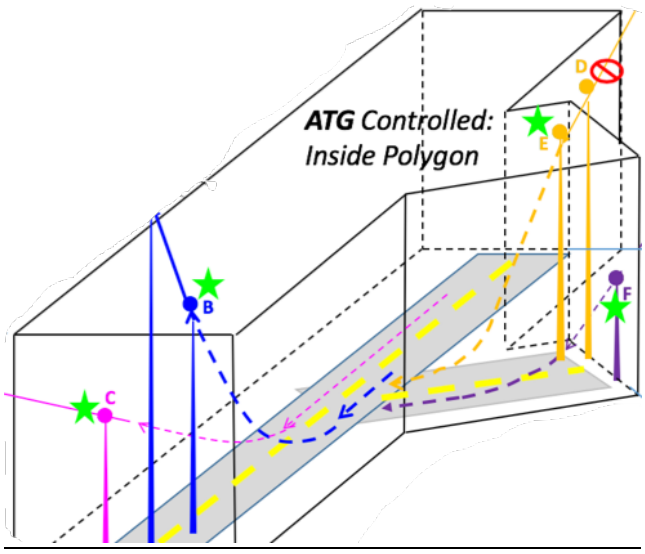

Fig. 4 - Example of the Handoff Between Target Generators

IV. Design Options

Although the combined ATG and MACS target generation system works well for the ATD-2 simulations, using both target generators require twice the number of pseudo-pilot workstations and participants for a typical ATM simulation, which has tremendous cost and logistical impacts. In addition, NASA researchers are anticipating that the planned UAM simulations will require an increase in the number of simulated aircraft, which may make this solution unfeasible. As such, the SimLabs development team is considering alternative solutions, each with their own pros and cons:

1.) **Continue the two-target generator paradigm**, developing additional compatibility between the two target generators, improving the hand-off mechanism, reducing the amount of overhead in the middleware, and attempting to scale up to UAM density levels. This still does not address the issue of requiring higher update rates, which inversely impacts the ability to scale.

2.) **Choose one target generator**, removing the necessity of handoffs, and attempting to scale up to UAM density levels and the required update rates for smooth out-the-window visuals. However, neither of the two target generators design architectures are optimized to take advantage of computers with increasingly higher number of cores and utilize parallel processing, but instead are designed to take advantage of processor speed.

3.) **Develop a new target generator**, leveraging the architectural benefits of clustered computing and parallelization to improve scalability and update rates. Where possible, it should cut out any high level of overhead that middleware solutions may impose on simulations by offering a standardized interface for all external services, such as the controller displays and pilot stations.

The SimLabs development team is in fact moving forward with all three of these options. The existing two-target generator solution works well for simulations of similar scope and airspace as the ATD-2 simulations and can be improved with limited system refactoring. SimLabs is also investigating potential mitigations for MACS and ATG limitations with respect to update rate and system scalability. The focus of this paper is the investigation and prototype development of a new target generator (Option #3).

If one were to develop a new target generator, one of the key technologies to consider is clustered computing and parallelization. Cluster computing is the method of utilizing multiple instances to provide additional resources for highly parallelized computations. By networking multiple computer systems and scheduling tasks in a controlled manner via a centralized server, a software system can benefit from scaling based on the number of nodes in a cluster. The nodes are managed using a specialized implementation of a load balancer, a system that can intelligently maintain proper scheduling between each of the nodes in a cluster and can spin-up additional nodes to help scale the system as demand increases. The scalability of clustering, when combined with parallelized computation, allows for greater

computational power and update rates through the principles of distributed computing. Major performance increases can be attained by utilizing clustered and parallelized computing for target generation.

When designing a prototype target generator, the features of programming languages desired were: first class parallelization, strict typing, and compiled binaries and cross-platform compilation, both desired for easy distribution. With these features in mind, languages like The Go programming language[9], C, and Java were considered. Go was ultimately chosen for the prototype due to its similarity to C, allowing for easy adoption, rapid developer velocity, and strong native parallelization. By using Go, each aircraft target can independently run its dynamics in its own micro-thread and be multiplexed onto the Operating System threads to highly parallelize execution and utilize multiple CPU cores. This removes the necessity to have one extremely powerful main system running an entire simulation.

To focus the target generator development objectives to meet NASA's research and simulation needs, and building upon lessons learned (and software components) from ATG and MACS, the following three primary tasks were identified to develop an improved target generation system prototype and test with a representative simulation scenario:

1.) Based upon the last-gen target generators, implement the dynamics code for the new stack, especially with non-traditional and futuristic air transportation operations in mind
2.) Architect a networking system to leverage clustered computing and parallelization
3.) Develop clients, such as controller and pilot stations, using previous target generators as templates

V. Target Generation Design

This section describes the output of the primary tasks to create the Target Generation System Prototype, called the Distributed Aircraft Simulation Engine (DASE). DASE is divided into four main components shown in Figure 5, namely the Simulation Server, the Processing Nodes, External Interface, and the External Services (e.g. Pilot Stations). These main components enable parallelization of workloads by separating the core target generation processing from the management of the virtual aircraft and interface to the user.

A. Processing Node

The Processing Nodes, as seen in Figure 5 as the "Node" boxes, contain the core aircraft and trajectory generation models. These are designed to be modular to allow for different types of models to be used and support extending the framework as new models are developed. Individual nodes are given multiple aircraft by the DASE server to process dynamics, act on input requests that would impact dynamics, and output the aircraft state data to the Simulation Server. Multiple processing nodes can be created to enable scalability of the system. Analysis of this scalability will be presented in Section VII.

B. External Server

The External Server, as seen in Figure 5 as the "External Server" box, provides the connectivity between the simulation and any external services. The external interface acts as the broker and message translator in order to allow for different services to utilize different connection types such as Hyper Text Transfer Protocol (HTTP) requests, websockets, etc., while maintaining a single type of messaging protocol internally. In addition, the external interface validates all external service message syntax in order to ensure that the Simulation Server does not receive invalid requests that may impact performance.

C. External Services

External Services, as seen in Figure 5 by the 4 dark blue layered boxes labeled "External Services," are any piece of software that interfaces with the simulation but that the simulation can function independently without. These can be pilot stations, conflict detection and resolution functions, autopiloting, or controller stations. As an example, pilot stations provide the interface between the simulation and the pilot (or algorithm) that is virtually flying the aircraft. Maneuver commands for the aircraft are input into the Pilot Station and then sent to the external interface.

A. Simulation Server

The Simulation Server, also known as the DASE Server, seen in Figure 5 by the light green box labeled as "DASE Server," acts as a broker and manager, overseeing all aspects of the simulation system. This includes managing the processing nodes and load balancing the nodes in order to ensure no node has an exceedingly high load compared to the other nodes. The Simulation Server also acts as the router for messages between the core target generation nodes and the external interface. For the prototype system, the Simulation Server provides the interface for controlling the

simulation runtime via pause, start, stop commands, as well as scenario and configuration selection. In the future, these functions could be provided by external services and managed by the simulation server.

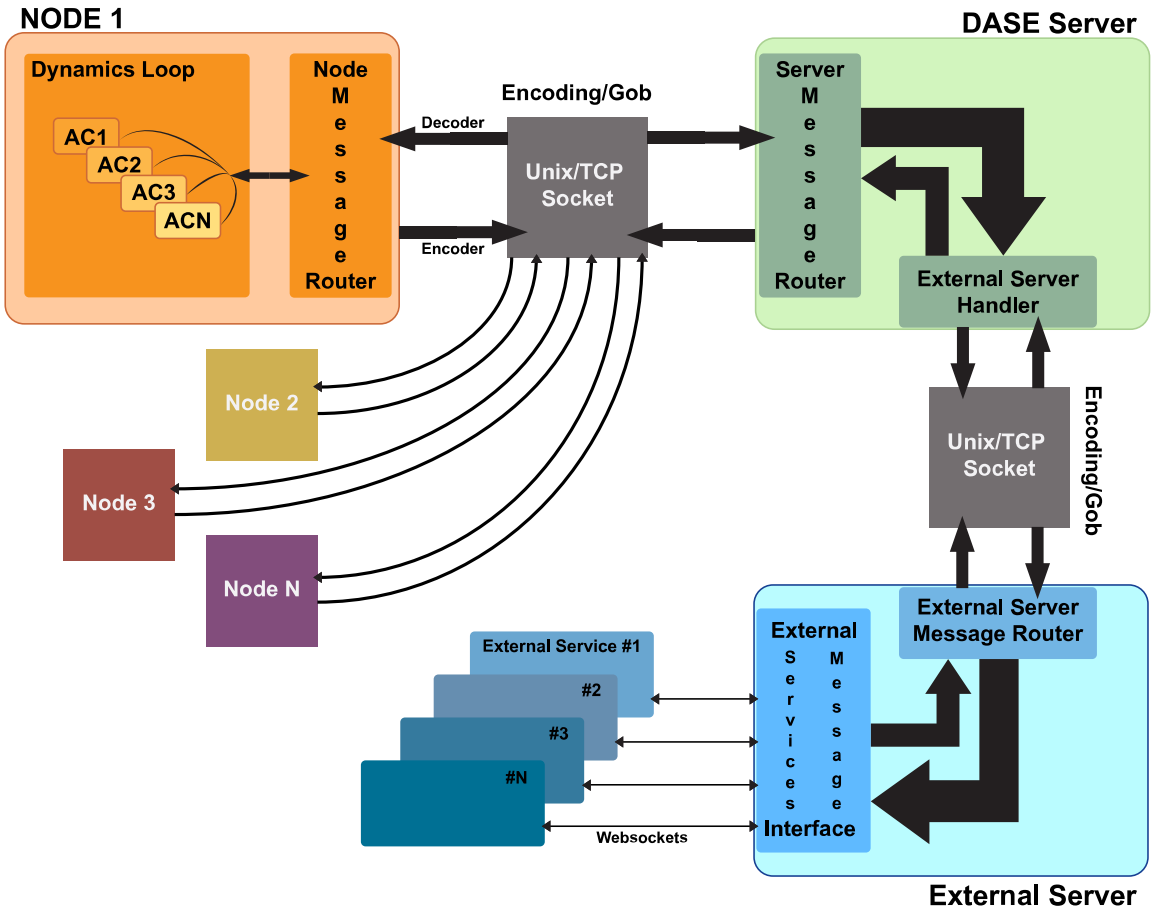

Fig. 5 - High level target generation system design.

VI. Performance Comparison of Serial, Parallel, and Distributed Processes in Computing

A. Computing Performance Description

Performance has become a major consideration in modern server infrastructure, with respect to serial (sequential, or single threaded), parallel (multi-threaded), and distributed computations. This is similarly a consideration in air traffic management simulations [10]. With the advent of modern web technologies, the utilization of parallelized workloads over distributed systems has increasingly become more important to scale technologies to the levels of network traffic seen in the modern web. The messaging density and scaling problems are very similar to that of a target generator in that sequentially processed events scale poorly with increased density. Parallelization provides additional scalability by offering multiple local instances of certain tasks in a computation pipeline, especially when calculations are independent of each other between parallelized threads. Additional threads do require additional memory, but can scale better than serial processing. Distributed processing allows the utilization of multiple hardware

systems to parallelize even further than a machine having multiple parallelized local instances. A graphical comparison of these computational methods can be seen in Figure 6.

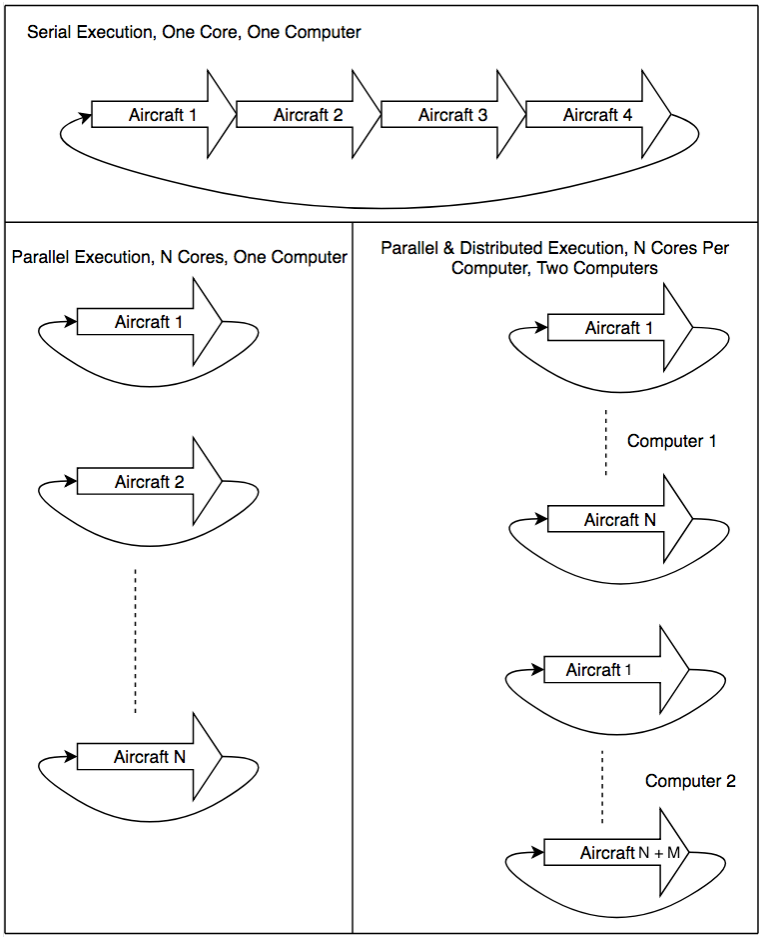

Fig. 6 - Serial, Parallel, and Distributed Parallel Execution

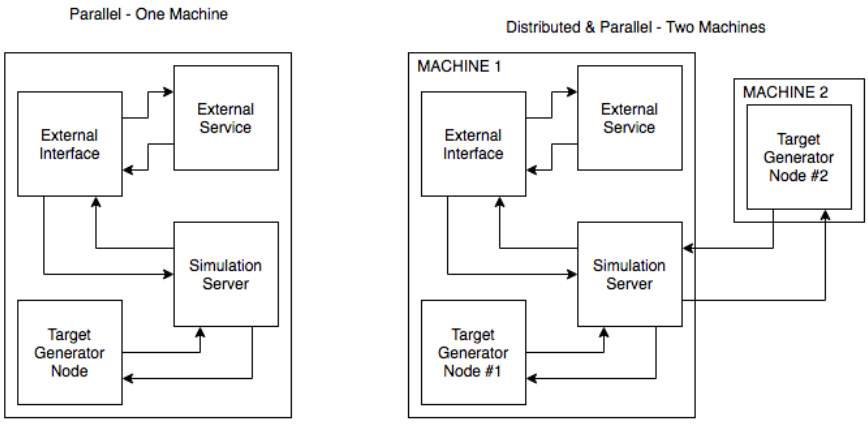

Fig. 7 - Parallel and Distributed system architectures.

B. Computing Performance Comparison

One key consideration is in optimization of the networking overhead between a single node and multiple nodes. Considering a computation, the total time to update all aircrafts' dynamics state once, with the total state update time *T*, where processing a single dynamics state for one aircraft takes time *t*, for *n* number of aircraft, using *m* number of machines, *p* number of threads per machine dedicated to updating aircraft states, and *C* the network overhead factor is described below for the three architectures:

$$T_{serial}(n,t) \sim (n \times t)$$

$$T_{parallel}(n,p,t) \sim \left(\frac{n}{p} \times t\right) \quad s.t. \;\; n \geq p$$

$$T_{distributed}(n,m,t,p) \sim \left(\frac{n}{p \times m} \times t\right) \times C \quad s.t. \;\; n \geq p \times m \;\&\; C < 1$$

These architectural descriptions illuminate the benefit of having distributed parallelized workloads. In addition, the importance of engineering the architecture to minimize the system and networking overhead is necessary to ensure scalability.

To compare serial, parallel and distributed architectures, a prototype system (as described in Figure 7) was developed. This prototype used a stand-in for a look-up trajectory model in which state-updates are generated through look-ups on performance tables. To isolate and uniformly benchmark the networking architecture performance, the stand-in operates via a standard library sleep function for a duration of time that emulates a look-up in a hash table and a three degree-of-freedom dynamics model calculation. The Target Generation system was benchmarked in two configurations, single system and dual system (See Figure 7). For the single configuration, we used one System76 workstation with 192GB of DDR4 RAM, two Intel Xeon Gold CPU's with a total of 72 threads, and running CentOS 7. Aircraft scenarios of 400 and 1000 aircraft were created to run through the systems to measure how well each system is able to scale. The primary focus of these analyses was the aircraft target update rate. Update rates of 60HZ or higher can be considered for high fidelity visuals for out-the-cockpit-window simulation. Rates above 4 Hz can be used for populating the visuals for out-the-window simulations at FFC.

C. Computing Performance Results

1. Summary of Performance Benchmarks

The summary of the test results in which one node and two node configurations were tested with 400 aircraft and 1000 aircraft scenarios is listed in Table 1.

Table 1. Summary of System Performance Results

Scenario Name	Number of Nodes	Number of Aircraft, Aircraft Per Node	Average Update Rate (Hz)
oneNode400 (Unix socket)	1	400, 400	125.4
twoNode400 (TCP socket)	2	400, 200	79.7
oneNode1000 (Unix socket)	1	1000,1000	48.9
twoNode1000 (TCP socket)	2	1000, 500	32.0

2. Single Node Parallel System Performance

This test uses a system on a single machine with a multithreaded workload with 400 aircraft. The results from five different test-runs is shown in Table 2. The average update rates for the aircraft targets all had low coefficients of variation, <1%, across each of the five data runs, more than twice the desired 60Hz rate, and more than four times the desired 30Hz for smooth visuals. The highest and lowest update rates are represented in bold text in the table.

Table 2. Single Node Parallel System Performance Results: 400 Aircraft

Single System (Unix)	Average Update Rate (Hz)	Std. Deviation (Hz)	Max Update Rate (Hz)	Min Update Rate (Hz)
Run 1	124.937	0.102	125.271	**124.819**
Run 2	125.252	0.071	125.494	125.165
Run 3	125.370	0.133	125.758	125.204
Run 4	125.320	0.095	125.562	125.201
Run 5	126.254	0.085	**126.503**	126.155
All Runs	**125.427**	**0.097**	**126.503**	**124.819**

When running the same system with 2.5 times the number of aircraft (1000), the outcome from 5 different test-runs shows a similar 2.5 times decrease in the target update rate (Table 3). The average update rates for the aircraft targets all had low coefficients of variation, <1%, across each of the five data runs, and exceeded 30Hz. The highest and lowest update rates are represented in bold text in the table.

Table 3. Single Node Parallel System Performance Results: 1000 Aircraft

Single System (Unix)	Average Update Rate (Hz)	Std. Deviation (Hz)	Max Update Rate (Hz)	Min Update Rate (Hz)
Run 1	48.867	0.192	**49.467**	48.659
Run 2	48.725	0.210	49.297	**48.503**
Run 3	48.778	0.187	49.299	48.579
Run 4	48.982	0.134	49.370	48.831
Run 5	48.997	0.142	49.419	48.839
All Runs	**48.870**	0.173	**49.467**	**48.503**

3. Distributed/Parallel System Performance Results

This test system demonstrates running multiple Target Generation Nodes on separate machines in a multithreaded workload, with 400 aircraft, 200 per system. The outcome from five different test-runs is shown in (Table 4). The highest and lowest update rates are represented in bold text in the table. As shown, the variability of the update rate is almost four times that from the single node test system, and the overall update rate is less than the single node system running on one machine. The reason for this is discussed in Section VII.

Table 4. Two Node Distributed/Parallel System Performance Results: 400 Aircraft

Dual System (TCP)	Average Update Rate (Hz)	Std. Deviation (Hz)	Max Update Rate (Hz)	Min Update Rate (Hz)
Run 1	80.346	0.671	**81.732**	**79.833**
Run 2	79.530	0.245	80.455	79.467
Run 3	79.314	0.238	80.079	79.339
Run 4	79.336	0.295	80.245	79.247
Run 5	79.226	0.458	80.416	78.883
All Runs	**79.550**	**0.381**	**81.732**	**78.883**

(Table 5) illustrates the results from the two node, distributed/parallel system test. As before, when running the same system with 2.5 times the number of aircraft (1000), the outcome from five different test-runs shows a similar 2.5 times decrease in the target update rate relative to the 400 aircraft case. Notice that the variability is still elevated compared to the single node case with 1000 aircraft, but is three times the amount of the 400 aircraft distributed system test. The reason for this is difference still under investigation.

Table 5. Two Node Distributed/Parallel System Performance Results: 1000 Aircraft

Dual System (TCP)	Average Update Rate (Hz)	Std. Deviation (Hz)	Max Update Rate (Hz)	Min Update Rate (Hz)
Run 1	31.955	0.164	32.415	31.698
Run 2	31.835	0.188	32.407	**31.518**

Run 3	31.933	0.199	32.614	31.638
Run 4	32.234	0.250	**32.971**	31.834
Run 5	32.119	0.216	32.811	31.778
All Runs	**32.014**	**0.203**	**32.971**	**31.518**

VII. Language Decisions and Benchmarking of Prototypes

One of the limitations of the parallelized architectures outlined above is the necessity for a programming language with a strong parallelization model. Standard operating system threads have a memory footprint on the order of one megabyte and do not support the amount of parallelization needed for each aircraft to have its own co-routine. Deterministic execution between co-routines is not necessary: if *aircraft A* finishes updating its aircraft state before *aircraft B* finishes updating its aircraft state this does not result in invalid execution, although it does violate determinism if *aircraft A* does not always precede *aircraft B*; however, each update on aircraft A must be sequentially executed to avoid leap-frogging. These requirements brought forward the necessity of a parallelization model that a language like The Go Programming Language (Go), supports. Go is expressive, concise, clean, and efficient. Its parallelization mechanisms make it easy to write programs that get the most out of multicore and networked machines, while its novel type system enables flexible and modular program construction. The language supports a high level of parallelization thanks to "goroutines," a lightweight thread-like routine that can take advantage of multiple CPU cores, and are multiplexed by the Go runtime scheduler into operating system threads. While having 100,000 or 1,000,000 operating system threads would lead to large-scale memory requirements – due to each thread requiring on-the-order of megabytes – each goroutine has a memory footprint on the order of single kilobytes. A simplified pseudo Go-like example of utilizing goroutines for a runtime-loop executing aircraft state updates is as follows:

```
for {
    for _, aircraft := range aircraftArray {
    go aircraft.UpdateAircraftStateLoop()
    }
}
```

For serial execution, the pseudocode is the same, sans the "go" keyword. For the goroutine version the completion of execution for each iteration is non-deterministic, as each iteration is independently spawned and concurrently executed – as outlined in the above paragraph – while in serial execution, the execution order is deterministic and sequential. Clustering provides the ability for scaling via increasing the number of machines available for processing, while parallelization provides the per-machine scaling. A prototype target generator was developed in which a constant-time, O(1), state-update stand-in was implemented using a standard library sleep function in Go based on projected expectations that, like other target generators, dynamics are based on a look-up table, which operates in O(1). The prototype uses the constant time state-update stand-in in order to specifically demonstrate scaling and increased fidelity based on the network and system architecture.

The single configuration also utilizes Unix domain sockets for the inter-process-communication protocol (IPC). For the dual system, two of these identical workstations were used, with the Transmission Control Protocol (TCP) as the IPC protocol. For the single configuration, all processes are run on a single system, including the processing node, simulation server, external interface, and an external service; in this case the external service is simply a websocket connection in Google Chrome that prints the state data into the Chrome javascript console. In the dual system configuration, one system is running the simulation server, the external server, an external service, and one processing node, while the second system is running only a single processing node. When the number of aircraft tested were increased in the single node configurations by two and a half times, a clear decrease in the average update rate by roughly two and a half times was observed. With two processing nodes the average update rate similarly scales in performance. However, when comparing the performance of the single node to the two node configurations, we see a one and a half times increase in performance due to a known discrepancy [7] between Unix Domain Sockets and TCP. When packets are larger than the ethernet packet size of 1500 bytes, a difference of fifty percent can be seen between the two protocols. This observed impact also increases as packet size and number of packets increases. However, when communicating between two computers Unix Domain Sockets, by design, are not usable.

VIII. Conclusion

In order to facilitate future air traffic management human-in-the-loop simulations which require greater scalability and update rate requirements than past, more traditional ATM simulations, NASA Ames Research Center's SimLabs

is evaluating enhancements to target generation systems. Technologies such as clustered and highly parallelized computing offer improvements to scalability and update rates for future simulations, such as urban air mobility which are anticipated to require modeling an order of magnitude more vehicles and smooth out-the-window visual systems. Testing and analysis demonstrated promise in the improvement and further development of potential target generators through the use of clustering and parallelization. The prototype provided update rates greater than 30Hz in all test conditions, and greater than 60Hz in the 400 aircraft test conditions. All tests had a low coefficient of variation, <1%, indicating stability. Moving forward, continued development of the prototype of such a system may provide tangible benefits in future simulations. Additional testing and development on the networking implementation in the prototype might yield a lower impact on performance due to TCP and allow for competitive scaling of nodes versus socket types. Relative to the high-end capabilities of the test systems, the networking performance acted as the limiting factor. Additional testing with multiple lower-end systems, such as single-board-computers, could yield comparable performance.

Acknowledgments

The authors of this paper would like to thank Carla Ingram, Karen Cate, Alan Lee, and Steven Beard for their expertise and feedback in the field of target generation at NASA Ames Research Center.

References

[1] Prevot, T., Smith, N., Palmer, E., Mercer, J., Lee, P., Homola, J., Callantine, T., "The Airspace Operations Laboratory (AOL) at NASA Ames Research Center," AIAA 2006-6112, AIAA Modeling and Simulation Technologies Conference, August 2006

[2] Lehmer, R. D., Malsom, S. J., "Distributed System Architecture in VAST-RT for Real-Time Airspace Simulation," AIAA 2004-5436, AIAA Modeling and Simulation Technologies Conference, August 2004

[3] Thipphavong, D., et al., "Urban Air Mobility Airspace Integration Concepts and Considerations," AIAA 2018-3676, Aviation Technology, Integration, Operations Conference, June 2018

[4] Weske, R. A., and Danek, G. L., "Pseudo Aircraft Systems: A Multi-Aircraft Simulation System for Air Traffic Control Research," AIAA-93-3585-CP, AIAA Flight Simulation Technologies Conference, August 1993

[5] Mercer, J., Prevot, T., Jacoby, R., Globus, A., and Homala, J., "Studying NextGen Concepts with the Multi-Aircraft Control System", AIAA 2008-7026, AIAA Modeling and Simulations Technologies Conference, August 2008

[6] NASA (2006) NASA Ames Simlabs URL: http://www.simlabs.arc.nasa.gov/index.html

[7] Pitch (2012) High Level Architecture URL: http://www.pitch.se/hlatutorial/

[8] NASA (2018) ATD-2 URL: https://www.aviationsystemsdivision.arc.nasa.gov/research/atd2/index.shtml

[9] Google (2018) Go Programming Language URL: https://golang.org/doc/

[10] Coa, Y., Sun, D., "A parallel computing framework for air traffic flow management," IEEE Transactions on Intelligent Transporation Systems, July 2011