18th International Conference on Space Operations, Montreal, Canada, 26 - 30 May 2025.

SpaceOps-2025, ID # 179

Simultaneous Constellation Maneuver Optimization using AWS

Ed Meletyan[a]

[a] *Amazon Web Services (AWS), United States*, *emelet@amazon.com*

Abstract

Cloud-based solutions offer new approaches for managing the growing complexity of satellite fleet operations. This paper presents a serverless architecture that leverages AWS Lambda, Amazon DynamoDB, and AWS Step Functions to perform collision avoidance analysis and maneuver planning for large satellite constellations. In a test scenario involving 5000 satellites and a debris field, the system identified 141 potential collisions and computed fuel-optimal avoidance maneuvers in 69.4 seconds - demonstrating a 579-fold improvement over traditional serial processing methods. The solution screens satellites for conjunctions in 42.5 seconds and computes maneuver plans in 26.9 seconds, with a total infrastructure cost of $0.25 per simulation. By distributing computations across thousands of parallel processes, the architecture enables rapid response to debris-generating events while maintaining high computational accuracy. This research validates the feasibility of serverless computing for space traffic management and provides a scalable framework for future constellation operations.

Keywords: mission planning constellation optimization cloud serverless

Nomenclature

$\Delta\vec{v}$	Impulse maneuver vector (m/s)
d	Distance between a satellite and debris field centroid (m)
$\vec{r}_{sat}$	Satellite position vector in inertial coordinates (m)
$\vec{r}_{debris}$	Debris centroid position vector in inertial coordinates (m)
$\vec{r}_{ca}$	Satellite position at the time of closest approach (m)
J	Cost function to minimize (m/s)
t_{wait}	Time to maintain original orbit trajectory prior to avoidance impulse (s)
t_{ca}	Time to closest approach (s)
t_{flight}	Time of flight from impulse to closest approach

Acronyms/Abbreviations

Amazon Web Services (AWS)
Conjunction Data Message (CDM)
Course of Action (COA)
Comma-Separated Values (CSV)
Identity and Access Management (IAM)
JavaScript Object Notation (JSON)
JavaScript Object Notation Lines (JSONL)
Low Earth Orbit (LEO)
Runge-Kutta 8(9) (RK89)
Satellite Toolkit with Rust (SatKit)
Simple Storage Service (S3)
Simplified General Perturbations 4 (SGP4)
Virtual Central Processing Unit (vCPU)

1. Introduction

The satellite industry has experienced unprecedented growth, with the number of active satellites expected to increase by 70,000 between 2025 and 2030 [1]. This growth has created new challenges for mission operators managing increasingly complex satellite fleets. Traditional approaches to conjunction assessment and maneuver planning, typically executed on local workstations or monolithic applications, cannot efficiently scale to meet these demands [2].

The proliferation of satellites in Low Earth Orbit (LEO) has intensified the computational requirements for collision avoidance. Operators must process thousands of potential conjunctions while maintaining rapid response capabilities for dynamic orbital conditions. Horizontal scaling of compute infrastructure enables parallel processing of mission planning activities, allowing operators to evaluate multiple courses of action within operational timelines.

This research addresses mission planning efficiency through parallel orbit propagation for conjunction assessment and distributed optimization of avoidance maneuvers. The solution delivers increased decision velocity through parallel processing, optimized cost-performance through pay-per-use computing, and simplified deployment and maintenance through serverless architecture.

1.1 Mission Planning Scenario

This research examines a critical operational scenario: rapid response to a debris-generating event affecting a large satellite constellation. When operators receive notification of a debris event through a CDM, they must quickly determine which constellation assets require maneuvers and compute fuel-optimal avoidance trajectories within the available response window. The scenario assumes accurate state vector knowledge for both the satellite constellation and debris field centroid, enabling immediate execution of the constellation maneuver optimization workflow.

1.2 Infrastructure Design

Modern cloud architectures enable "serverless" computing, where infrastructure scales automatically with demand. This approach provides mission planners with on-demand parallel processing of thousands of scenarios through a pay-per-use pricing model for compute and storage. AWS manages the underlying infrastructure, handling scaling, patching, and availability, allowing operators to focus on mission planning rather than computing resources. This fully managed service model eliminates infrastructure maintenance burden while ensuring computational resources are available when needed for time-critical operations.

2. Material and methods

This research integrates mission planning tools with cloud infrastructure to enable fleet-wide collision avoidance maneuvers. The architecture orchestrates these tools through cloud services to deliver maneuver solutions to operators in near real-time.

2.1 Mission Planning Components

The solution incorporates three primary mission planning capabilities: orbit propagation, maneuver planning, and maneuver optimization. Orbit propagation and analysis rely on the open-source SatKit library [3], though the architecture supports integration with any validated flight dynamics software. This toolkit provides high-performance orbital kinematics calculations, coordinate system transformations, and time system management.

Maneuver planning employs a Lambert solver based on Oldenhuis's implementation [4], optimized for Python 3.12. This solver determines the optimal orbital transfer trajectory between two position vectors within a specified time constraint. The implementation handles both short-way and long-way transfers while maintaining numerical stability.

The solution implements a genetic algorithm for maneuver optimization, configured to balance solution quality with compute time constraints. The algorithm evolves a population of 50 candidate solutions over 100 generations, typically converging to fuel-efficient maneuvers within 30 seconds. This approach provides robust solutions while meeting operational timeline requirements.

2.2 Cloud Infrastructure

The serverless architecture comprises five integrated AWS services. Amazon S3 provides the foundation for data management, storing function dependencies, input state vectors, and execution logs with 99.999999999% durability. Its integration with other cloud services and cost-performance makes it ideal for this application.

Amazon DynamoDB serves as the primary operational database, storing ephemeris data and maneuver plans as key-value pairs. Its consistent single-digit millisecond performance and automatic scaling capabilities support thousands of simultaneous write operations from parallel processes.

AWS Lambda forms the computational core, executing orbital mechanics calculations with high concurrency. Each satellite's analysis runs in an isolated container with up to 10GB of memory and 6 vCPUs, enabling parallel processing of up to 10,000 simultaneous instances. AWS Lambda's support for multiple programming languages allows direct integration of existing mission planning software.

AWS Step Functions orchestrates the workflow, managing the distributed execution of AWS Lambda functions through state machines. It handles parallel processing coordination, error recovery, and execution monitoring. The service automatically scales based on input workload while maintaining execution history for audit purposes.

AWS Identity and Access Management (IAM) provides security controls across all components, ensuring proper access management and resource isolation.

2.3 Operational Workflow

The solution implements a two-phase approach to collision avoidance planning. The first phase performs conjunction assessment through parallel orbit propagation and collision detection. AWS Step Functions initiates this phase by processing input state files (supporting CSV, JSON, or JSONL formats) and distributing propagation tasks across AWS Lambda functions. Each function analyzes its assigned satellite's trajectory against the debris field, writing collision parameters to Amazon DynamoDB when detecting potential conjunctions.

The second phase focuses on maneuver planning for identified collision risks. AWS Step Functions queries Amazon DynamoDB for at-risk assets and launches parallel AWS Lambda functions to compute avoidance maneuvers. Each function executes the genetic algorithm optimization to determine fuel-efficient maneuvers while meeting timing constraints. The resulting maneuver plans, including impulse vectors and execution timing, are stored in Amazon DynamoDB for operator review. The solution architecture diagram is shown in Figure 1.

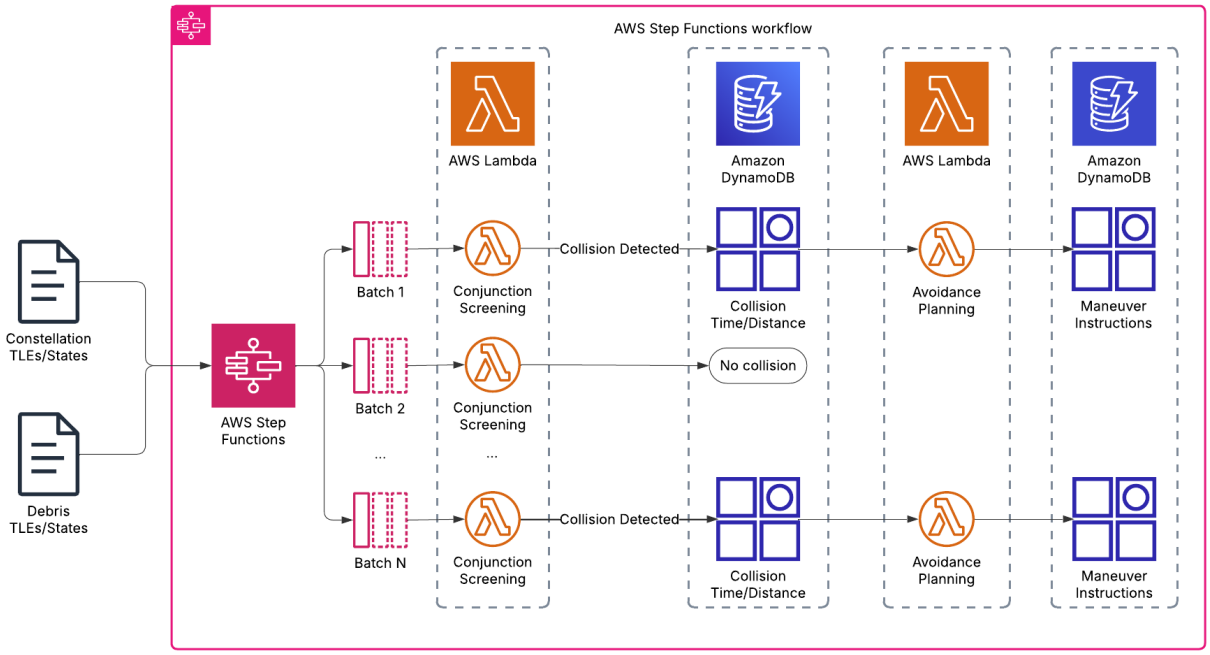

Figure 1. AWS architecture for constellation conjunction screening and maneuver planning.

The infrastructure automatically scales to support up to 10,000 parallel AWS Lambda instance, with each instance utilizing up to 6 cores. This scaling capability, combined with Amazon DynamoDB's elastic storage, eliminates the need for advance capacity planning while maintaining consistent performance under varying workloads.

3. Theory and Calculation

The solution integrates orbital mechanics principles with distributed computing architecture to enable scalable conjunction assessment and maneuver planning. This section details the mathematical foundations and computational methods underlying the system.

3.1 Cloud Service Integration Architecture

The distributed storage and compute model optimizes performance for orbital calculations through a dual-storage approach. Amazon DynamoDB provides rapid access to ephemeris and maneuver data through a partition key design based on satellite unique identifiers. This design ensures consistent single-digit millisecond access times while supporting atomic updates through optimistic concurrency control. Each database record contains a unique satellite identifier and associated message payload. The payload structure encompasses timestamp data, collision indices, and detailed maneuver plans including wait duration, maneuver duration, and three-dimensional Δv components.

The compute architecture leverages AWS Lambda's isolated execution environments, each preloaded with orbital dynamics libraries. These environments access configuration parameters through environment variables defining mission-specific constants such as Earth's physical parameters and the debris keep-out radius.

3.2 Conjunction Assessment Methodology

The conjunction assessment phase implements a distributed simulation for collision probability calculation. The process begins with orbit creation from stored orbital elements, followed by propagation using RK89. The propagation fidelity is tunable by the user including analytic, semi-analytic, and numeric techniques. This propagation occurs across specified time intervals, accounting for perturbations.

The collision detection algorithm computes the minimum approach distance between each satellite and the debris field through vector analysis. For each time step, the algorithm calculates:

$$d = |\vec{r}_{sat} - \vec{r}_{debris}| \tag{1}$$

When d falls below the specified $r_{keepout}$, the system records the conjunction in Amazon DynamoDB along with the time of closest approach and relative geometry. Figure 2 shows an example of a collision detected by the algorithm.

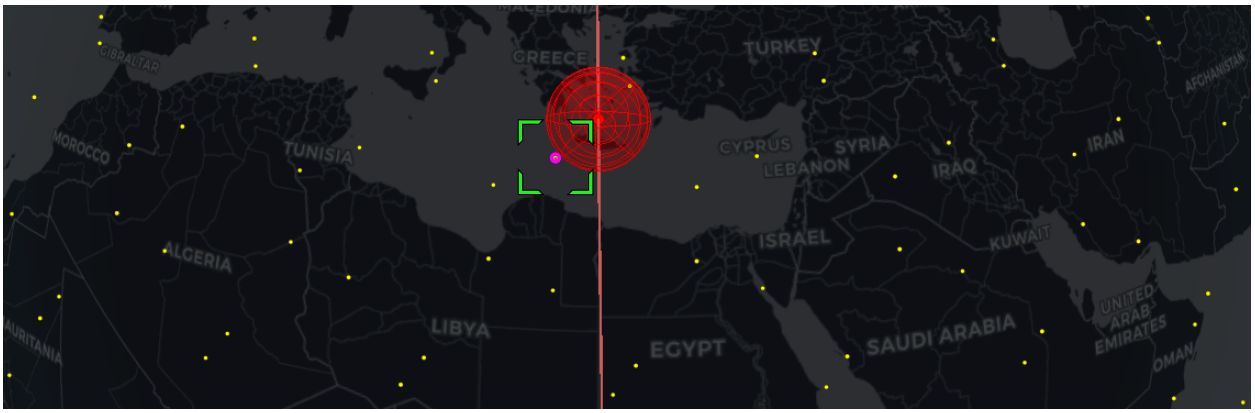

Figure 2. Nadir view of a satellite (yellow) approaching the keep-out zone of a debris object (red).

The calculation first determines the satellite position at the time of closest approach to the debris $\vec{r}_{ca}$, then generates a safety position $\vec{r}_{safe}$ according to:

$$\vec{r}_{ca} = (\vec{r}_{sat})_{t=ca} \tag{2}$$

$$\vec{r}_{safe} = \vec{r}_{ca} * SF \tag{3}$$

This safety position incorporates a predetermined safety factor SF to ensure adequate separation between space objects. The safety position is restricted to the satellites current orbit plane to reduce cross-track maneuver components that would change its inclination.

3.3 Maneuver Optimization Framework

The maneuver optimization employs a genetic algorithm to minimize the cost function J:

$$J = |\Delta\vec{v}| \tag{4}$$

The algorithm searches for impulse and flight times within the bounds:

$$0 \leq t_{wait} \leq 0.5 * t_{ca} \tag{5}$$

$$t_{flight} = t_{ca} - t_{wait} \tag{6}$$

For each candidate solution, the Lambert solver computes the required velocity change vectors. Given position after t_{wait}, target position $\vec{r}_{safe}$, and time of flight t_{flight}, the solver determines the orbital transfer trajectory minimizing the total Δv magnitude.

The genetic algorithm maintains population diversity through tournament selection and uniform crossover operations. Mutation operations apply Gaussian perturbations to maneuver timing parameters, with standard deviation scaled to the remaining search space. The optimization terminates upon reaching either convergence criteria or the maximum generation count. The maneuver strategy is illustrated in Figure 3.

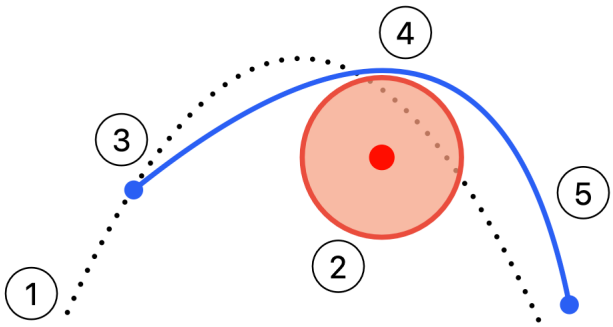

Figure 3. Collision avoidance strategy. 1: Initial orbit, 2:Debris keep-out zone at time of closest approach, 3:Maneuver start, 4:Target safety position, 5: New trajectory

The solution provides the necessary velocity vectors, from which the required Δv components are calculated by comparing the Lambert solution velocity with the initial orbital velocity at the time of the impulse.

The system stores the resulting maneuver plans in Amazon DynamoDB using a structured format that includes wait duration, flight duration, and three-dimensional Δv components. Figure 4 shows an example of an Amazon DynamoDB item containing the maneuver plan for a specific spacecraft (denoted by the 'uid' field).

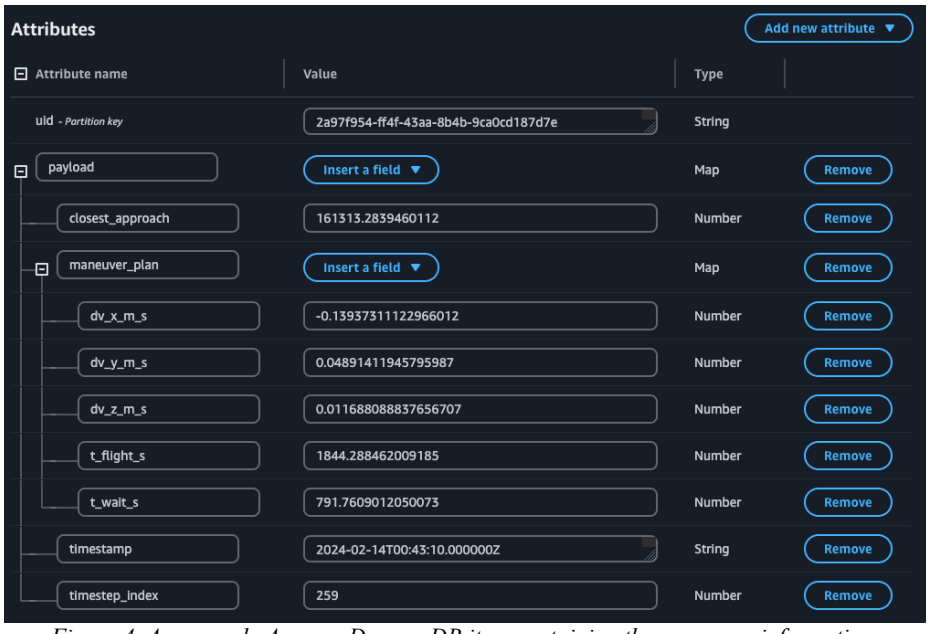

Figure 4. An example Amazon DynamoDB item containing the maneuver information.

The attributes in each maneuver plan item are:

- 'uid' (unique spacecraft identifier)
- 'payload' (representing a grouping of attributes)
  - 'closest_approach' (distance in meters of the closest approach between the satellite and debris)

- ‘maneuver_plan’ (a grouping of maneuver plan components)
  - 3-dimensional impulse vector in dv_[x/y/z]_m_s in meters/second
  - t_flight_s representing the time in seconds to reach the safety position after the impulse is applied
  - t_wait_s representing the time in seconds that the satellite should stay on its current trajectory prior to applying the impulse
- ‘timestamp’ (timestamp of the closest approach)
- ‘timestamp_index’ (index of the timestamp in the context of the analysis window)

The persistent storage in Amazon DynamoDB ensures that maneuver plans remain accessible for operator review and execution while maintaining data consistency across the distributed system. Amazon DynamoDB can also trigger follow-on workflows whenever an item is created or updated. These workflows can include converting the maneuver plan to a spacecraft command, performing additional validations, or notifying an operator.

4. Results and Discussion

This section presents the performance analysis of the serverless architecture applied to a large-scale constellation management scenario. The test case comprised 5000 satellites in circular orbits at 1050 km altitude and 63.5-degree inclination, distributed uniformly in right ascension and true anomaly. Figure 5 shows the orbital geometry at the start of the scenario.

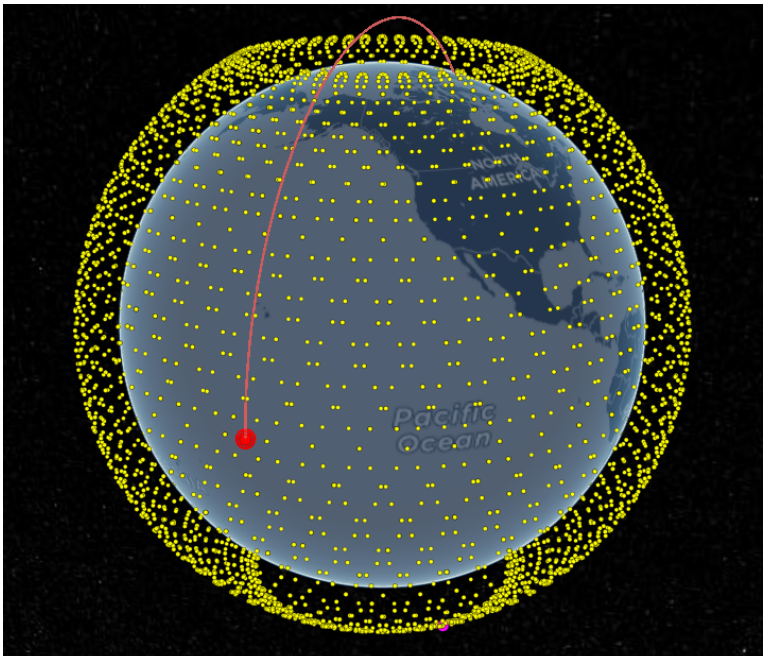

Figure 5. Orbital geometry of test scenario with maneuvering satellites (yellow) and debris trajectory (red).

The constellation encounters a debris field modeled as a 200-kilometer radius sphere in a polar orbit at the same altitude, with the analysis spanning a one-hour operational window.

4.1 System Performance Analysis

The distributed processing architecture achieved significant computational efficiency through parallel execution. The conjunction assessment phase completed in 42.5 seconds, while maneuver optimization required 26.9 seconds, yielding a total execution time of 69.4 seconds. AWS Step Functions contributed minimal overhead, averaging 0.05 seconds per state transition (5 total in this solution). Compared to traditional serial execution methods, which would require approximately 11.1 hours (10.6 hours for conjunction assessment, 0.5 hours for maneuver optimization), the parallel architecture achieved a 579-fold reduction in processing time.

AWS Step Function provides an intuitive interface to create state machines that orchestrate distributed functions. It reduces the operational burden of maintaining complex workflows and maintains a detailed audit of previous executions and their results.

4.2 Collision Detection Effectiveness

The simulation approach identified potential collisions with high accuracy. The system processed orbital states for both constellation satellites and debris objects, evaluating multiple time steps for each potential conjunction. The distributed processing capability enabled simultaneous evaluation of numerous scenarios, with Amazon DynamoDB maintaining consistent state information throughout the execution. Over an hour of simulation time, the solution detected 141 collisions for a fleet of 5000 satellites (2.8%).

The number of collisions detected is dependent on the time step used in propagation as large time steps (greater than 20 seconds) can miss detections. However, choosing a time step that is too low (1 second) can unnecessarily inflate the computational burden. The step size can be configured and should be carefully evaluated in an operational scenario. To improve this solution, it is possible to add interpolation to the closest approach calculation such that it is only performed when the closest approach is under a threshold.

4.3 Maneuver Optimization Results

The genetic algorithm optimization produced efficient avoidance maneuvers for identified conjunction events. The Lambert solver successfully generated viable trajectories while minimizing Δv requirements. The system computed three-dimensional maneuver components for each identified collision risk, storing the results in Amazon DynamoDB for operator review. Figure 6 shows the distribution of maneuver magnitudes against the time of flight available starting from detection of debris and collision.

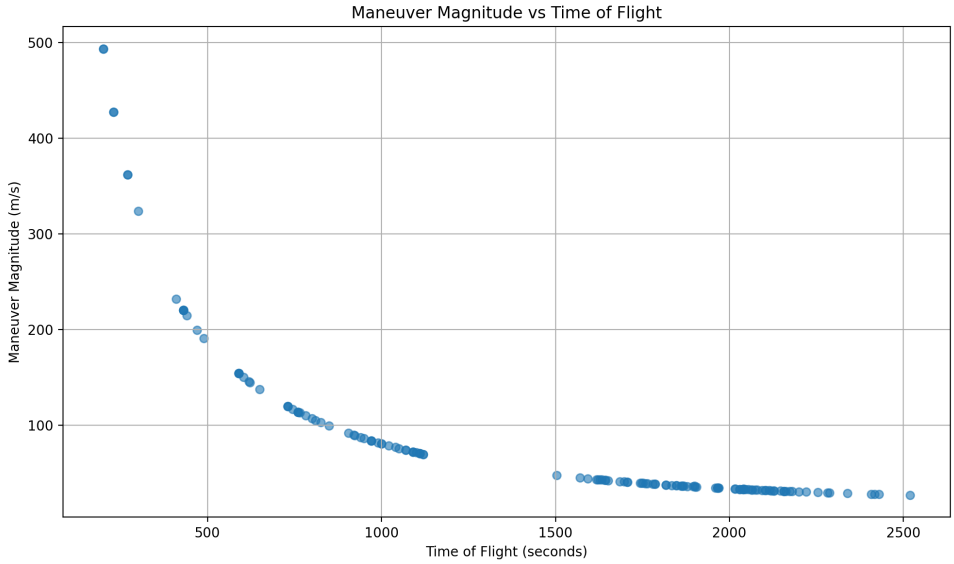

Figure 6. Plot showing distribution of computed Δv values across all maneuver solutions.

The exponential increase of fuel required to maneuver to safety as the remaining time to collision decreases highlights the importance of detecting debris and reacting to it as quickly as possible. It should be noted that there is a gap between 1200 and 1500 seconds. This gap is due to the lack of collision risk as the debris approaches the North pole and the constellation is inclined to 63.5 degrees.

4.4 Computational Performance

The serverless architecture demonstrated significant advantages in processing speed and resource utilization. The system efficiently scaled to handle varying workloads, with AWS Lambda functions executing in parallel to process multiple conjunction scenarios simultaneously. Table 1 outlines the duration statistics of the individual AWS Lambda functions that AWS Step Functions launches (up to 10,000 in parallel).

Table 1. Summary of AWS Lambda function run durations

Function	**Minimum Duration (s)**	**Average Duration (s)**	**Maximum Duration (s)**
Collision screening	0.2	7.6	26.2
Maneuver optimization	10.3	12.4	13.6

One primary benefit of using AWS Lambda for this experiment is the service's flexibility. AWS Lambda can run containers under 10GB in size and functions written a wide range of languages (Node, Python, Java, .NET, Ruby, Rust). Users can run the tools of their choice at scale with this service.

4.5 Database Performance

Amazon DynamoDB maintained consistent performance throughout the execution, with an average of 18 ms access times for write operations and 29.8 ms for scan operations. The storage system effectively managed the high volume of state data and maneuver plans, with no observed performance degradation during peak processing periods.

As with AWS Lambda and AWS Step Functions, Amazon DynamoDB is a fully managed cloud service which means the infrastructure provider handles scaling, patching, and state of health of the service. As a result, data can be continuously written to the database from multiple AWS Lambda sources without the risk of running out of storage.

4.6 Cost Analysis

The serverless architecture demonstrated cost efficiency through its pay-per-use model. AWS Lambda function execution costs remained proportional to the actual computational workload, while Amazon DynamoDB costs scaled with the volume of stored data and access patterns. This resulted in predictable operational costs that scaled linearly with constellation size. Using the AWS Pricing Calculator [5], the total cost of the experiment run in this study is $0.25. Table 2 outlines the costs in more detail.

Table 2. Cost estimate of a simulation

Function	Configuration	Cost per 1 hour simulation
AWS Lambda	Invoke Mode (Buffered) Architecture (x86) Number of requests (5141 per execution) Amount of ephemeral storage allocated (512 MB)	$0.24
Amazon DynamoDB	Table class (Standard) Average item size (20 Byte) Data storage size (0.001 GB)	$0.01
AWS Step Functions	Duration of each workflow (6900 ms) Memory consumed by each workflow (64 MB) Workflow requests (1 per execution)	$0.00

The primary benefit of serverless is that users do not need to provision a large amount of compute to handle simulations that happen as a reaction to singular events or do not need to be running constantly at maximum capacity. In this experiment, the solution provisions 30,000 virtual cores for 42 seconds and then gracefully shuts the resources down. The user is billed only for the duration the compute is active meaning that it is not necessary to maintain physical servers that provide the same amount of compute when they are only used during a debris-generating event detection.

4.7 System Reliability

The solution demonstrated robust error handling and recovery capabilities. The AWS Step Functions workflow successfully managed retry logic for failed computations, maintaining the 1% failure tolerance threshold for the collision detection phase. The system achieved a 100% success rate in completing full workflow executions, with automated error recovery handling the remaining cases. Figure 7 shows the output of an AWS Step Functions distributed map run for collision avoidance screening.

Figure 7. Example processing workflow reliability of AWS Step Functions.

AWS Step Functions provides options for re-driving failed events as well if the individual AWS Lambda processes return errors. Users can also define custom workflows to handle the failures such as notifying an operator or sending the failure to a separate log.

5. Conclusions

This research demonstrates the effectiveness of leveraging cloud-based serverless architecture for large-scale satellite constellation management. The solution successfully processed collision avoidance scenarios for 5,000 satellites, identifying 141 potential collisions and computing optimal maneuvers in just 69.4 seconds - a 579-fold improvement over serial execution methods. The serverless architecture processed conjunction assessments in an average of 7.6 seconds per satellite and maneuver optimizations in 12.4 seconds per satellite with AWS Lambda functions. The system's ability to parallelize computations across thousands of instances demonstrates its suitability for large constellation operations. The pay-per-use model of serverless computing resulted in a total operational cost of $0.25 per simulation run, making it an economically viable solution for both regular operations and emergency response scenarios. The system achieved 100% workflow completion with built-in error handling and recovery capabilities, demonstrating the robustness required for mission-critical applications. Amazon DynamoDB maintained consistent performance with 18 ms write times and 29.8 ms scan times, even during peak processing periods with multiple simultaneous operations. The genetic algorithm successfully generated fuel-efficient avoidance maneuvers, with results demonstrating the relationship between available response time and required delta-v magnitude which necessitate rapid detection and response to debris-generating events in LEO.

The results validate the feasibility of using cloud-based serverless computing for complex space traffic management tasks. This approach provides satellite operators with a scalable, cost-effective solution that can respond rapidly to debris-generating events with the user’s preferred astrodynamics library.

Future work should focus on incorporating machine learning models for predictive conjunction assessment, expanding the system to handle inter-constellation coordination, and developing enhanced optimization algorithms for multi-satellite coordinated maneuvers. Collision avoidance maneuver planning can be improved as well by incorporating optimal maneuver strategies that look at out-of-plane maneuvers. The architecture presented here provides a strong foundation for these advanced capabilities while maintaining the benefits of serverless computing's scalability and cost-effectiveness.

The significance of this research extends beyond immediate operational benefits, suggesting a paradigm shift in how complex space traffic management computations can be handled using modern cloud infrastructure. This approach may become increasingly valuable as the number of satellites in orbit continues to grow, requiring more sophisticated and scalable solutions.

References

[1] Goldman Sachs, "The global satellite market is forecast to become seven times bigger," 5 March 2025. [Online]. Available: https://www.goldmansachs.com/insights/articles/the-global-satellite-market-is-forecast-to-become-seven-times-bigger.

[2] D. Garner, "The Limitations of Monolithic Ground Segments for Mission Management," Cognitive Space, 18 April 2024. [Online]. Available: https://www.cognitivespace.com/blog/limitations-of-monolithic-ground-segments/. [Accessed 4 April 2025].

[3] S. Michael, "Satellite Toolkit with Rust," https://github.com/ssmichael1/satkit. [Accessed 4 April 2025].

[4] R. Oldenhuis, "Robust solver for Lambert's orbital-boundary value problem," https://github.com/rodyo/FEX-Lambert/releases/tag/v1.4. [Accessed 4 April 2025].

[5] Amazon Web Services, "AWS Pricing Calculator," https://calculator.aws/#/. [Accessed 4 April 2025].