DISTRIBUTION OF CONFLICT DETECTION OF AIRCRAFT FOR NEXT GENERATION FLIGHT MANAGEMENT SYSTEMS

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Scott Kuroda

June 2013

COMMITTEE MEMBERSHIP

TITLE:	Distribution of Conflict Detection of Aircraft for Next Generation Flight Management Systems
AUTHOR:	Scott Kuroda
DATE SUBMITTED:	June 2013
COMMITTEE CHAIR:	Dr. Franz Kurfess, Professor of Computer Science
COMMITTEE MEMBER:	Dr. Phillip Nico, Associate Professor of Computer Science
COMMITTEE MEMBER:	Dr. Alex Dekhtyar, Associate Professor of Computer Science

ABSTRACT

Distribution of Conflict Detection of Aircraft for Next Generation Flight Management Systems

Scott Kuroda

As the number of aircraft is expected to triple in the coming decades, the manual process used to safely route aircraft while in flight will become insufficient. There already exist work to algorithmically detect safe and unsafe routes between aircraft. This thesis extends that system such that the computation is distributed across multiple machines. In addition it also supports the detection of an unsafe route as it is actively modified by a third party. Furthermore, the system supports providing safe or unsafe route notification to multiple interested clients.

ACKNOWLEDGMENTS

Thank you the NASA Flight Deck Display Research Lab for the support through this project.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Over the past century, the advancement in commercial air travel has made flying much more affordable. The affordability of air travel led to an increase in demand for air travel services. To cope with the increase in demand, additional aircraft were produced. This increase in aircraft placed additional strain on the air traffic control systems. A similar scenario is likely to occur in the coming decades. With this in mind, enhancements to the traffic control systems are continuously being researched.

While in flight, an aircraft is required to maintain a defined amount of separation between themselves and other aircraft. The separation required between aircraft is defined by the aviation authorities. Maintaining this separation minimizes the risk of in air collisions between aircraft. If two aircraft violate this separation space, a loss of separation is said to occur. At this point, additional safety systems will activate to prevent collision between aircraft.

If a path is defined, such that two aircraft will have a loss of separation, these aircraft are said to be in conflict. Ideally, routes would be defined ahead of time

such that conflicts do not occur. However, due to the unpredictable nature of some factors of air travel, such as weather, routes are often modified while an aircraft is in flight. Because of this, care must be taken to minimize conflicts while rerouting aircraft. If a route is defined such that a conflict does occur, a resolution must be found as quickly as possible.

The management and subsequent resolution of conflicts falls to two parties. The first is air traffic control. Air traffic control is a ground-based service that, among other important tasks, manage aircraft routes in a designated space. The second is the flight crew. The term "flight crew" refers to those responsible for the operation of the aircraft while it is in flight. This includes the flight attendants, captain, first officer, etc. The safety of the aircraft and any passengers are the responsibility of those on the flight deck. Primary responsibility falls onto the pilot. If something was onto happen to the pilot, the responsibility would fall to the first officer.

Today's method for managing aircraft is largely manual, relying on the abilities of air traffic controllers to actively organize aircraft in an assigned region. As such, it is not something that scales well as the number of aircraft in a region increases. In order to maintain the high level of safety expected during air travel, it is important that these types of systems are able to scale along with the number of aircraft.

To help improve the efficiency of managing aircraft routes while in flight, aircraft systems are being enhanced with modern technology. In order to leverage these new systems, the traffic management services must also be updated.

The work presented improves upon next generation flight control systems. The work is based on a system being developed by the NASA Flight Deck Display Research Lab (FDDRL) [16]. Much of their work focuses on incorporating flight crews in a more significant roll in route management tasks. This work addresses a number of issues present in the system in development at the FDDRL. The high level goal is to create a distributed computational system to detect conflicts, thus allowing the system to handle an increased number of aircraft simultaneously. The system will accept data from multiple data sources. In addition, it allows multiple multiple clients to request conflict data from the system simultaneously. Clients may also request flight data from the system.

The remainder of the work will be structured as follows. Section 2 will discuss related work. The current system developed by FDDRL will be discussed in Section 3. The description of the project will be described in Section 4. The design and implementation of an system to address the limitations will be discussed in Section 5. Section 6 will present an evaluation of the system. Sections 7 and 8 will discuss future work and concluding remarks respectively.

Chapter 2

Related Work

The work done on this project requires an understanding of two fields, air traffic management and Computer Science. An expert understanding of air traffic management is not required, rather a general understanding of the limitations. In addition, because this project builds off of technology that is already categorized as next generation, an understanding of what challenges it is already overcoming is also important. Because of the vast difference between the fields, they will be discussed in turn, beginning with air traffic management aspects, then the Computer Science aspects.

2.1 Air Traffic Management

As the goal of this project is to improve upon air traffic management tools while aircraft are in flight, it is important to have a basic understanding of today's systems, as well as the next generation systems this project builds off of. These are discussed in the following sections.

2.1.1 Today's Systems

Air traffic control today utilizes radar for locating aircraft in a given region of space. In addition, secondary radar systems are used to communicate with transponders aboard aircraft. These transponders provide various kinds of information when requested by the secondary radar systems. As radar based systems are based on bouncing a signal off of aircraft, there is some delay in the data used by air traffic control. Though the delay is manageable, having more accurate real time data would allow air traffic controllers to provide safer instructions to aircraft [25].

When loss of separation occurs, the traffic collision avoidance system (TCAS) becomes active. Whereas the radar based systems provide information to air traffic controllers, the TCAS system, provides information at a much shorter range to the surrounding aircraft. This exemplifies the usefulness of real time data. The importance in a loss of separation scenario is clear, however, it would also be useful to air traffic management [7].

The communication system between aircraft and air traffic control is audio based. The radio communication allows pilots to hear communication with other aircraft. Because of this, they can gauge how busy an air traffic controller is. However, this also points to a limitation with the communication method. With enough traffic, the audio channel could become so congested that it is difficult for pilots and air traffic control to communicate. This presents the need for additional communication methods, especially as the number of aircraft increases.

There are, of course, other technologies, but these are the key components of managing aircraft while they are in flight. Though the amount of traffic is manageable today, a mostly manual process is not something that scales well. As the amount of traffic is expected to triple by 2025 [23], the current process must be updated. The updated system must support the increased air traffic expected in the coming decades.

2.1.2 Next Generation Systems

The next generation systems, some of which are being implemented now, address some of the issues discussed in 2.1.1. Of particular interest are the data communication methods and Automatic Dependent Surveillance - Broadcast (ADS-B).

The data communication methods are used to compliment the audio based communications used today. As previously mentioned, the audio based channels provide valuable insight into the load an air traffic controller currently has. Though any use of data communication methods will reduce the accuracy of this analysis, it still provides an additional means for an air traffic controller and pilot to communicate.

ADS-B is a system that provides similar information as the radar systems used by air traffic controllers. Rather than relying on the "ping" of a radar, an aircraft will now broadcast information about its state. Some examples of state information are velocity, heading, latitude, and longitude. The information that is broadcast will then be received by ground based stations. These stations will then transfer information to air traffic control, or any relevant parties connected to the network. In addition, these broadcast will communicate data to nearby aircraft.

In a sense, this is similar to the real time information shared by TCAS. However, it operates over greater distances. This system is not without limitations. As it relies on information being broadcast from an aircraft, there is some potential for an aircraft to not broadcast its state or broadcast incorrect information. In addition, there is a possibility for external systems to spoof an aircraft. Because of these limitations, the ADS-B system will be used in conjunction with radar based systems.

Both of these components are a part of the "Next Generation Air Traffic Systems" (NextGen). NextGen is considered to be the first major update in air traffic management in the last half century [17].

With ADS-B pilots would have access to similar information of that of air traffic controllers with regards to the state of surrounding air craft. Because of this, some tasks previously done by air traffic controllers can be delegated to pilots. One such task is defining new routes to manage various events that occur while in flight. This is not a new concept as it was previously presented in [2, 3]. However, this is not as simple as allowing pilots to define and act on their own routes. Air traffic controllers must still be aware of the actions taken by pilots. A number of studies have taken place to determine how distributed air traffic management can be incorporated into the system. These studies focus on how the new systems will affect air traffic controllers and pilots. In addition, the affect on the awareness of air traffic controllers is also analyzed as the systems are meant to support air traffic controllers, not replace them [9, 29, 32].

Before discussing the actual distribution of work, knowledge with respect to how the information is displayed is important. To display the information provided by existing, as well as newly developed systems, a new display system is needed. For traffic-specific information, the FDDRL developed the the Cockpit

Display of Traffic Information (CDTI). This system was also the basis for the Cockpit Situation Display (CSD) developed by the FDDRL [1, 16]. The CSD displays a variety of information beyond that of the two dimensional information provided today. An example of the CSD display can be seen in Figure 2.1. The details of the current CSD system are discussed in Section 3

Figure 2.1: Image of 3D CSD [11].

2.2 Distributed Computing

A distributed computational system can provide excellent scalability as the number of computations increases. As such, its development continues to be a popular area of research. Though there are many key points in designing a distributed system, key underlying concepts are architecture, fault management, and load balancing.

2.2.1 Architecture

Although research in this area continues to improve on existing design patterns, two common architectures for distributed computing are master/slave and peer-to-peer.

In a master/slave configuration, there is a single machine that hands out work to the slave machines. By forcing all communication through the master, work can often be distributed in efficient ways up front. In addition, it is easier to ensure that all slaves have the data they need to perform there computational task. However, having a master can produce a single point of failure. This can be avoided by designing a system such that another machine can act as master if a failure occurs [28].

Peer-to-peer networks do not have a hierarchical structure like a master/slave solution. Instead, all workers are considered equal. These types of systems can be more complicated than a master/slave solution. This is because each client must be able to communicate with every other client. Whereas a master/slave configuration allows each worker to simply do the work they are assigned, a peer-to-peer system may need to pass work from one client to another. It may also need to request data from another client to complete its computation [27].

2.2.2 Fault Management

Fault tolerance is a key component in distributed systems. Faults can come from a variety of sources. Some common faults are the failure of computational nodes and network failures. When faults like these occur, it is important that the distributed system is still able to continue its work. Though there is a number

of ways to detect faults in a distributed system, one common method relies on a heartbeat-like ping. When a response is received, a node is considered still active. Though there are other methods to detect faults, this particular method lends itself well to this project [8, 30].

When a fault occurs, there is generally data that is lost. How a distributed system handles this loss of data varies from application to application.

In an application that processes static data, the system will often perform the lost computations again. This would produce accurate and consistent results. However, there is an additional cost in performing computations again. The time it takes to redistribute work and perform a computation again negatively impacts the overall runtime of the system. For applications that process static data, this trade-off is often acceptable.

If instead an application is processing real time data, it may not be necessary to redo lost work. By the time the lost work is redistributed and recomputed, the data used to perform the computations may already be considered out of date. In a scenario like this, it may be acceptable to lose some data accuracy, to maintain a higher overall throughput.

A given application may utilize a combination of these methods. In a certain state, it may require redoing computations. Whereas another state may allow for less accurate data to be used. The exact implementation for an application must take into account the effects of lost accuracy and the decrease in overall throughput.

2.2.3 Load Balancing

For any type of distributed application, load balancing is key to producing the highest possible performance. If done incorrectly, computational resources will be underutilized, degrading the performance of the overall system. As with most problems, there is no single solution that is best in all scenarios, leading to the development of a variety of load balancing methods.

Depending on the underlying system architecture, a load balancer may be classified as either centralized or decentralized. A centralized load balancer routes all work through a single master node. This node, generally classified as middleware, is responsible for distributing work to the rest of the compute nodes. While this can produce better results, as the master node should be aware of who work has been given to, and who should get more work, it can be fragile. Because all of this communication is occurring through a single point, this becomes a single point of failure. This can be mitigated by designing the system with a type of fail over protocol [22].

Decentralized systems require all of the compute nodes to talk to each other. If one node feels it has to much work, it can pass it off to another node. Similarly, if a node has too little work, it may request additional work. This type of setup is ideal in peer-to-peer distributed computing systems [15].

Moving beyond the underlying architecture, a load balancer may be either static or dynamic. Both have a set of cost and benefits, which must be analyzed when determining what is best for a given application.

Static methods are often simpler as they rely on some predefined knowledge of the compute nodes. They often have a much lower overhead cost when assigning work as there is minimal analysis performed when passing work to compute nodes. However, these often underutilize the resources available to the system [18].

Two examples of static distribution methods are round-robin and random distribution. Round-robin distribution ensures that no new work is given to a compute node until all of the other compute nodes have also been assigned work. This type of balancing works well in a system where all of the compute nodes have equivalent compute power. In addition, the jobs assigned to each machine must take a similar amount of time to finish. Random distribution assigns work to workers in a random fashion. With a good random function, this would distribute work in an even way. However, it is still possible that less computationally powerful machines are given more work, leaving other machines idle while work is being completed [24].

Dynamic distribution methods, in some sense, are an enhancement of static methods. In addition to using predefined information, dynamic distribution also incorporates runtime data in the distribution process. Types of runtime data that are commonly used are processor usage, memory usage, IO usage, and network latency. Using all this additional information to intelligently distribute work is not without its cost. It relies on accurate data being reported to the load balancer. In addition, the time it takes to assign work with this intelligence is often greater than that of a static method [14]. Some examples of dynamic load balancing are fuzzy biasing, geographic partitioning, and scatter distribution.

Fuzzy biasing is a centralized distributed computing system. It assigns tasks to workers, who perform a single task at a given time. Rather than using specific details such as memory and processor consumption, this load balancer uses the number of tasks assigned to a node and the amount of time the previously completed task had to wait to be executed. With these pieces of information for each node, a score is generated. The load balancer then assigns work based on this score [26].

The geographic partitioning algorithm is a method developed to perform traffic simulations in New York, New York [31]. This type of method is very specific to distributed computations that are running to support some sort of simulation. These simulations should involve the location of entities in some simulated space. In the geographic partitioning method, the simulated space is divided into a grid. A computational node is then assigned some number of grid spaces. This node is responsible for all of the computations that need to be performed on entities within that space. Using simple rectangular spaces quickly proves to be inefficient as some grid spaces can require significantly more computations than others. This leads to an uneven distribution of overall work. To overcome this, nonuniform grid spaces can be used. While this does allow for a more even distribution than the rectangular grids, it can still produce an uneven distribution of work. Another potential optimization to this involves performing load prediction. The load predictions can be used to temporarily reassign work or redefine the grid boundaries. However, there is a significant amount of overhead in doing this. In addition, it requires advanced knowledge of the cost of tasks that are going to be performed. Though in some scenarios this can produce positive results, the additional overhead upfront can outweigh the benefits seen in distributing the work in a more intelligent manner [12].

Also presented in [31] is scatter distribution. In geographic partitioning, a computational node was responsible for all of the computations in a given region. Put another way, a computational node is responsible for all of the computations associated with an entity in a given region. In the work presented in [31], an entity would be a vehicle. As an entity moves from grid space to grid space, the computational node responsible for performing the computations for that entity changes. In high traffic areas, the number of entities in a grid space can be significantly higher than that of the surrounding grid spaces. This leads to an uneven division of work. Scatter distribution works to overcome this limitation. Rather than assigning a computational node to a set of grid spaces, the computational nodes are assigned a set of entities. In scatter distribution, a computational node is responsible for performing all of the computations associated with a given entity. Even within a high traffic region, the work is distributed across multiple machines, leading to a much more balanced overall work load.

As an example, consider scenario with 1500 entities. Furthermore, 1000 of them are concentrated in a single geographical region, as defined by the geographic partitioning previously discussed. This means that a single computational node performs computations for 1000 entities, while other computational nodes manage the remaining 500. In scatter distribution, all of the entities or divided up equally among all of the computational nodes, resulting in a more even distribution of work. Of course, this is not without its drawbacks. Because an arbitrary node is responsible for a given entity, it is the only node that has accurate information about that entity. When computations occur, this data must be shared with the other computational nodes. This leads to a much higher communication cost prior to performing any actual computations. By dividing work at the entity level rather than at a grid level, it can also be much easier

to move work from one compute node to another. This distribution method is again specific to simulation like computations. Whether it is better or worse for a given simulation will change from application to application.

Chapter 3

Current System Design

The Cockpit Situational Display (CSD) system is built in such a way that components can easily be added and removed, modifying the overall functionality of an individual display. For the purpose of this project, there are two key components the conflict detection and resolution (CDnR) system and the route assessment tool (RAT). The rest of the CSD is treated as a black box. The black box component is responsible for passing messages to the CDnR system, displaying alert and traffic data, and passing updated route information from the RAT to the CDnR system. The RAT is a component of the CSD, but is not directly modified by this work. Because this project focuses on extending prior work, much of the information derived about the current system is a result of exploring the system. The architecture of the current system can be seen in Figure 3.1.

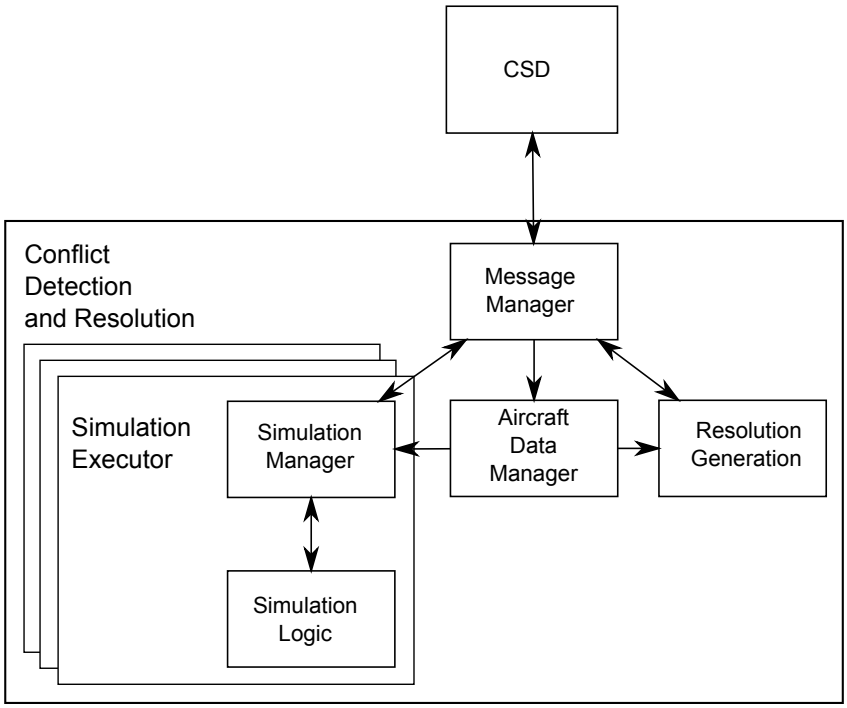

Figure 3.1: Current System Architecture

3.1 Conflict Detection and Resolution

The CDnR system consists of three main components: message management, conflict detection, and resolution generation. For the purpose of this project, the key components are message management and conflict detection. Therefore, they are the focus of the following discussion [5].

3.1.1 Message Management

The message management component is responsible for relaying information to and from internal threads and external processes. An important type of message being passed to this component from an external source is flight information. These messages contain information about aircraft flight plan and aircraft state.

This information is used by the conflict detection and resolution components. In addition, an external process must specify the ownship aircraft. This ownship aircraft refers to the aircraft of interest. For a pilot, the aircraft they are piloting is their ownship. This is important as what computations the conflict detection and resolution system performs is based on the ownship specified.

Messages may also be received specifying if a conflict detection simulation should be run. This component manages starting simulations, pausing simulations, and stopping simulations. The number of conflict detection simulations that can run is limited by an enumeration, detailing the types of routes that can be specified. In its capacity as manager, it also accepts a number of configurations used to modify the behavior of the simulations. As an example, an external process may specify pre and post filtering to be done on the simulation. The preprocessing filters will limit the data sent to the simulation, using the information of aircraft state. The postprocessing filters will limit the alerts returned, based on the contents of the alerts. Another piece of configuration is specifying how often the simulation should run. Additionally, the configurations specify the vertical and horizontal region to be used in determining if a conflict occurs. These are the key components of the configuration used by the conflict detection simulation. There are others that are specific to the simulation method. Details of the simulation method and associated configurations are discussed in Section 3.1.2.

In addition to receiving messages from external processes, this component also receives a number of messages from internal threads. Of particular interest is the conflict detection alerts. When it receives these alerts, it forwards them to the process that requested the simulation to be run. In addition, based on the route information being used in the simulation, active or modified, some additional work may be done. As an example, if a conflict exist on an active route, the resolution service will be started, to assist in quickly determining a safe updated route.

3.1.2 Conflict Detection

The conflict detection component performs the actual simulation that generates alert data. Each simulation is responsible for generating its own data, based on the information gathered by the message manager. The current conflict detection system uses a Monte Carlo algorithm to perform the conflict detection simulations. Configurations are passed from the message manager to a conflict detection thread when a simulation is requested. In addition to the previous items mentioned, the configuration also contains a maximum rate at which the simulations should run. In addition the configurations also contain algorithm specific information. In this case, an oversampling rate to be used by the Monte Carlo method. The oversampling rate specifies how many samples should be taken before a cycle is deemed complete. If incomplete, the conflict detection system reuses the data from the previous iteration, rather than generating new data. If instead a simulation cycle is classified as complete, this component gathers the aircraft data and generates a set of data to be used by the simulation algorithm. After a cycle is completed, the results are passed back to the message management component [4].

Figure 3.2: Image of CSD displaying conflict between two aircraft [13].

3.2 Route Assessment Tool

The RAT component allows for a pilot to define a new route. When activated, this system displays the current route with the way points currently defined along the route. The pilot may use an existing way point, or select an arbitrary point on the path to create one. They can then reposition the way points, producing a new route. These routes are sent to the conflict detection system. Doing this ensures that there are no new conflicts along the newly defined route. However, it is possible to define and accept a route that has conflicts. Though a real world implementation will likely pass these results to air traffic controllers for final validation, the current system immediately accepts the new route. If a conflict exist along the newly defined route, an alert is presented to the pilot [6].

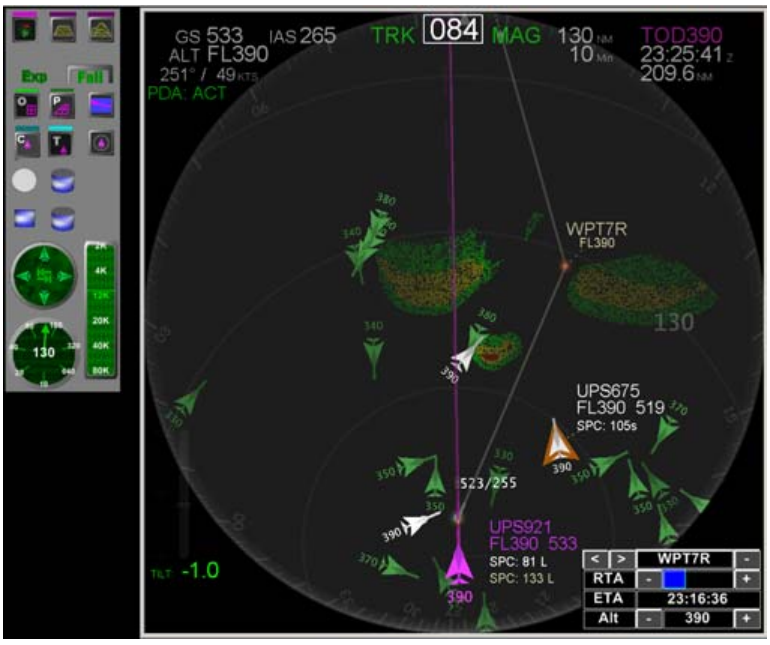

Figure 3.3: Image of CSD with RAT active [10].

3.3 Limitations

Although there already exists some work to show conflicts to pilots, it is not without limitations. Many of these limitations are a result of the original design in which the conflict detection system would be run locally for each aircraft.

The original system was designed to be run on each aircraft. This means that each aircraft manages its own conflict computations. This makes it difficult to share results in real time. Though this is not really an issue with identifying conflicts on active routes as all aircraft should have a similar data set, it does present a problem when proposing a new route. It may be useful to relay information about a proposed route in real time to a party beyond that of the current aircraft. For example, while making a routing decision, air traffic controllers may want to actively participate, rather than simply receiving a route proposal.

Second, assuming a point is reached in which data is being shared, it is likely that a single system would be responsible for computing conflicts for multiple ownship aircraft. With this in mind, the computational limitations of a single machine system may be reached quickly. Theoretically speaking, a worst case scenario would involve running conflict detection using every aircraft as an ownship aircraft. So, given N aircraft, there would potentially be N^2 computations to be done using the active data. This is, of course, a worst case scenario as filtering could be applied to aircraft outside of some given range where conflicts are not calculated. In addition, for active cases, half of the computations done would be repeated. Thus, optimizations can be made to reduce the overall number of computations. This holds true for computations being done on modified routes proposed by the RAT system. Even with the filtering and other optimizations that could occur, a point would be reached in which a single machine is unable to perform the computations for the given number of aircraft.

Chapter 4

Problem Definition

The goal of this project is to address the limitations in the current system as described in Section 3.3. Specifically, this project produces a proof of concept system that allows for the results of the conflict detection simulation to be shared with multiple clients.

4.1 System Requirements

The system can be broken into client and server components. To address the limitations of the previous work, the system design shall meet the following criteria.

4.1.1 Clients

The system shall support three main types of clients: data clients, worker clients, and results clients. In addition, there shall be no hard limit on the number of clients that can connect to the server.

Data Clients

Data clients shall provide aircraft data or configuration data to the server. The aircraft data shall be a minimal set of information needed to perform conflict detection simulations. The configuration data shall specify simulation specific settings to be used when checking for conflicts.

Worker Clients

The worker clients shall execute the conflict detection simulations. The conflict detection algorithm used is based on previous work done by FDDRL. In addition, each worker client shall be able to run multiple simulations simultaneously to help improve overall performance.

Results Clients

The results clients shall receive information on conflicts for a given aircraft. In addition, results clients shall be able to request aircraft data used to perform a given simulation.

4.1.2 Server

In addition to supporting the requirements of the clients as previously specified, the server shall control the execution of the conflict detection service. As with the current system, the maximum rate at which simulations are run shall be controlled by a configuration passed to the system. The simulation service itself shall support multiple types of simulations simultaneously. Each simulation shall have an independent set of configurations, which may be specified by a configuration client.

4.2 Integration

This proof of concept system shall be tightly integrated into the current CDnR and CSD systems. Each CSD shall be set up to provide data to the system and receive results. The results generated should be similar in both the distributed system presented and the current non-distributed system. In addition, each simulation iteration should be completed within a reasonable amount of time, such that the results can be used in a real time scenario.

Chapter 5

Design and Implementation

The high-level design of the system consists of a server and multiple types of clients, as seen in Figure 5.1. The computational aspect of the system can be described as a master/slave system, where a single machine delegates work to the worker machines. In addition to generating work for slave machines, the server will also be responsible for managing a number of data collection and result client connections. Each of these clients is discussed in turn, including their integration into the current system.

5.1 Clients

There are four types of clients supported in this application. They are the data gatherer, configuration, worker, and results clients. Each will be discussed in turn, as they they are all important for a fully implemented system.

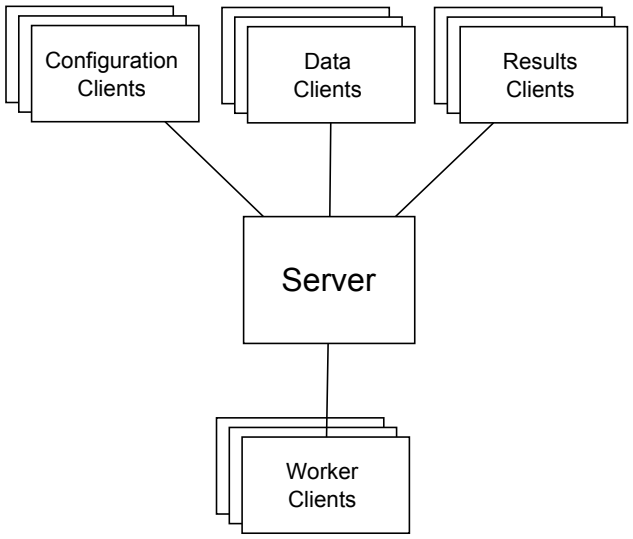

Figure 5.1: High level architecture of designed system.

5.1.1 Data Gathering Client

The data gathering client is responsible for providing data to the server. When compared to the current implementation, this client handles the aircraft state and aircraft route messages. In addition, it also passes a value specifying if alerts should be generated for the modified route.

Sending aircraft data to the server should produce a consistent amount of traffic. However, it does not have to. In the case it does not, it is important to ensure the connection is still active. This is done through a heartbeat, which is sent after a predefined amount of time. Under the current design, there is no additional information sent with this heartbeat message. Because this client is simply a data provider, it is unlikely that any other information will be supplied to the server. However, this functionality can be expanded upon.

5.1.2 Configuration Client

The configuration client passes configuration data to the simulation system. This configuration data can specify information such as simulation period and filtering methods. While being integrated into the current system, each CSD will have a configuration client. However, future work will likely remove this, moving the management of the configurations to a client independent of an individual aircraft.

5.1.3 Worker Client

The worker client runs the actual simulation. It is very similar to the conflict detection threads in the current system. The architecture, seen in Figure 5.2, consist of a connection, a simulation manager, and a simulation executor.

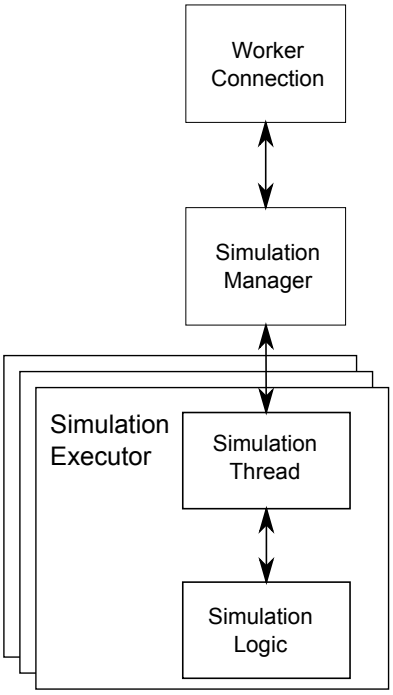

Figure 5.2: Architecture of worker client.

The connection is the clients interface to external processes. It accepts flight data, simulation configuration, and details on what simulations to run. The details of simulations to run come into the system as a series of pairs containing aircraft ids. One of the id's specifies the ownship aircraft and what route to use, modified or active. The second is the aircraft to run the simulation with. In addition, the connection provides information back to the server.

The simulation manager provides data to the simulation executors, similar to the message manager of the current system. However, this passes the actual data to be used by the simulations rather than forcing the simulations to generate the data themselves. It also gathers the results as the simulations complete, then makes them accessible to the connection so they may be passed back to the server.

Work begins when a "begin work" message is received by the connection. Following that message, are details about how the cycle should be run. Recall that if it is a new cycle or a continuing cycle as defined by the oversampling rate, the behavior will differ.

The flow will be further discussed in the following sections. An overview of it can be seen in Figure 5.3.

New Cycle

At the start of a new cycle, the worker will accept any new data. This data includes aircraft pairs to execute the simulations on, what route to use for the ownship aircraft for each pair, and the route data. The generation of the pairs and route data will be discussed in more detail in Section 5.2.3. After receiving this data, the worker takes all of the pair information and creates a complete list of aircraft that is to be compared against an individual ownship

aircraft. This generated list details what aircraft are to be compared against each ownship aircraft. After the list is generated, a second list is created, which contains all of the necessary route data to perform the simulation. If a simulation had not previously existed for a given ownship aircraft, it is started. If it did previously exist, the route information is updated and the simulation is run. After completing this run, the alert data generated for each aircraft is stored in a map structure and returned to the server.

Depending on the system configurations, it is possible that a worker may receive multiple request during the same cycle. This is an effect of trying to evenly distribute work. Because of the design of the conflict detection algorithm, this can result in repeat computations. Repeat computations occur if a particular worker receives a new computational pair where the ownship is an aircraft that had been executed in a previous run. This can be mitigated by adjusting how work is distributed by the simulation process.

Continuing Cycle

If this is the continuation of a previous simulation cycle, the same data is used to rerun the simulation. This is required by the design of the conflict detection algorithm. This does provide some benefits as new data does not have to be resent every computational cycle.

As with the data collection client, the worker client also sends heartbeats. However, it only actively sends them when it is performing computations. If it is not performing computations, it is simply responding to a heartbeat message sent by the server. Again, there is no additional information being sent back when sending the message to the server. However, this client may eventually send back information about its current state, allowing for better load management.

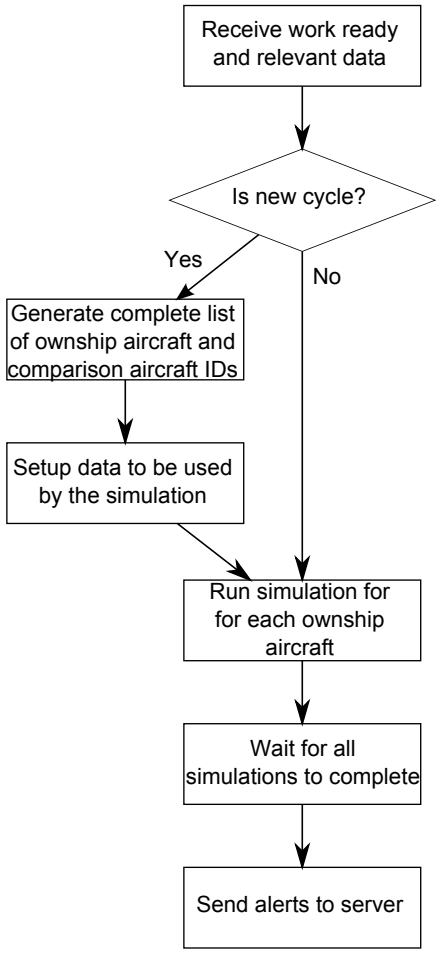

Figure 5.3: Worker client flow.

5.1.4 Results Client

The results client receives alert data after a simulation has been completed. Before that occurs, it must notify the server of the aircrafts it is interested in. This occurs when the server sends a heartbeat to the client. In most instances, an acknowledge message will be sent back to the server. However, it can also specify the aircraft the client is interested in.

The results client may request alerts for any number of aircraft at a given time. The results client is free to change the aircraft it is interested at any given time. However, results are currently only generated for requested aircraft, so there may be some additional delay when results are actually received. The alerts the client receives consist of data for both active and modified routes of the aircraft of interest. However, this does not mean both modified simulations and active simulations were run. It is possible for a particular entry to contain no alerts. This occurs when no modified route is being specified.

After receiving the alert data, the results client may optionally request the flight data used to generate the alert. This is requested by specifying the ids of the aircraft whose data is desired. The server then returns the flight plan and state data for the requested aircraft. This may be useful for a system that does not provide data, but would like to do some processing on the alerts. The flow for this client can be seen in Figure 5.4.

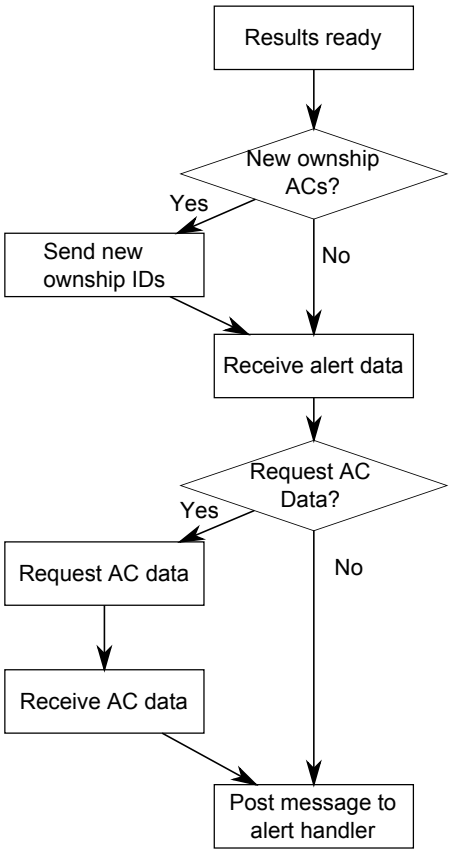

Figure 5.4: Results client flow.

5.2 Server

As previously mentioned, the server handles all of the connections from the various clients. Upon receiving a connection, this component creates a new connection handler thread. The specific handler executed is based on the type of connection, specified by the client. In addition, the server is responsible for the execution of the simulation process. The architecture of the server can be seen in Figure 5.5

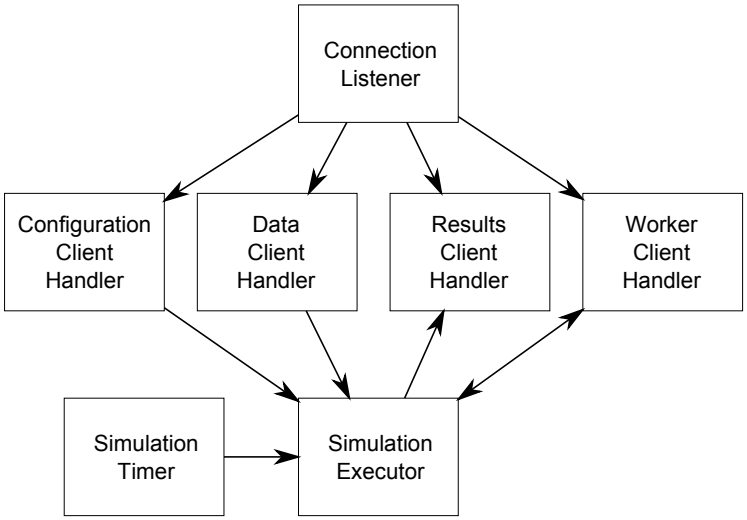

Figure 5.5: Internal structure of the server.

5.2.1 Configuration and Data Gathering Handlers

The configuration and data gathering handlers are quite simple. Upon receiving data, these handlers place the data in a local store. The location of each store is kept in a list that is accessed by the simulation process. The data that comes in is assumed to be timestamped. Though this is irrelevant to the handler itself as it simply replace old data as it is received, the simulation process does take the timestamps into account.

5.2.2 Results Handler

Prior to sending any results to a client, the client first specifies what aircraft it is interested in. It does this, as previously described, by sending aircraft identities to the server. Upon receiving these, the results handler attempts to create an entry in a results request list. This list is a map of aircraft ids to an integer value, representing number of clients requesting data. If an entry already exists, it simply increases the count. In addition, if the client had previously requested other aircraft, those entries are decremented. The results request list is accessed by the simulation process to determine what simulations to run.

5.2.3 Simulation Process

The simulation process runs in a separate thread that is started when the server begins. In addition, to throttle the rate at which simulation cycles are run, a timer is also started. The period of this timer is based on a configuration which was previously discussed. When the timer expires, it signals the simulation to begin.

As this builds off the current system, the simulations still use the Monte Carlo simulation previously created. Each simulation cycle may be classified as either new or a continuation. This determination is based on the oversampling rate specified by the configurations previously discussed. Depending on what type of cycle it is, the steps taken diverge. An overview of the flow can be seen in Figure 5.6.

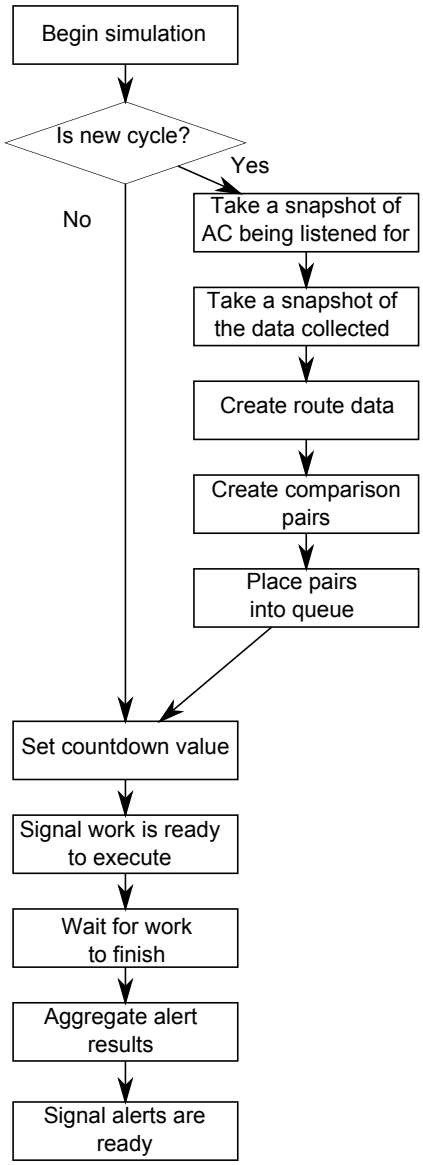

Figure 5.6: Simulation flow on server.

New Cycle

If it is a new cycle, the process first takes a snapshot of the aircraft being requested by results client. This snapshot data is produced by data entered by the results handlers, as specified in 5.2.2. This snapshot provides the ids of the aircraft being requested by the results clients. This is an optimization for the simulation process. First, it ensures that simulations are not run if no one is listening for results. Second, it ensures that only requested aircraft simulations are run. However, because of quick skip logic, there is a small amount of delay between when a results client request data, and when it receives it. The actual amount of delay seen by a results client is managed by two properties. The first is the maximum rate the simulation runs. If the simulation is set to run once every second, there will be, at most, a one second delay. The second property is how often the results handler provides a heartbeat to the results client. Recall that the results client may send an updated aircraft list as the response to a heartbeat.

If there are aircraft being requested, the process continues by taking a snapshot of the aircraft data. Recall that the simulation process, as described 5.2.1, has access to all of the data received thus far by the various handlers. The classes containing the data used when taking snapshots have been written in such a way to minimize the lock time. This is done to minimize the amount of time data is not being received and processed by the configuration and data gathering handlers. When generating the snapshot, the timestamps are used to ensure that only the most recent data is returned to the simulation process.

After this snapshot of the data is taken, a series of aircraft routes are generated. These routes are based on the data gathered as part of the snapshot. The data gathered contains more information than is actually needed by the simulation process. This evaluation creates a much smaller set of data that is needed for the simulation process. This evaluation is the most computationally intensive task performed by the simulation process. Though this could be moved to each individual worker, there would be a large number of redundant computations as each worker would have to recreate the same data. In addition, doing this on the server minimizes the amount of data that must travel across the wire to a worker client. After generating the route data, it is placed into two maps, one for active route data and one for modified route data. These two maps have keys that uniquely correlate to each aircraft.

Next, aircraft pairs are generated. This was previously touched on in Section 5.1.3. The pairs of aircraft are how work is being distributed to the worker clients. Each pair consists of an ownship id and a compare id. In addition, the ownship id entry contains a specifier telling the worker if that ownship id represents an active route or modified route. The pairs are generated by taking the active aircraft ids requested by the results client as the first entry and adding in every other aircraft as the second aircraft id. A similar process is done using the aircraft ids that have modified routes specified. While generating these pairs, a small amount of filtering is done to reduce the number of overall computations to be done.

After generating the the pairs, they are placed onto a queue accessible to the worker connection handlers. In addition, in order to prevent polling of the state of the queue, a countdown is created. This is set to the number of entries in the queue. This causes the simulation execution thread to block until the countdown reaches zero. The countdown reaching zero signifies that all work has been completed. After setting the countdown latch, the simulation process may modify the number of entries taken off the queue by the worker threads. The worker are then signaled that there is work to be executed.

When the work is completed, the executor gathers the results generated by the workers. Because the results for a single aircraft may have been computed on a different worker, the executor aggregates the results into complete sets. After this is done, the alert data, along with the flight plan and state data used to in the simulation, are placed into a shared object. This shared object is accessible by the results handlers. This shared object contains a cyclic buffer of results to help manage any potential delays that may result from an unreliable network. In addition, this object contains a read/write lock, allowing for multiple results clients to read simultaneously, while still blocking when an update occurs. Finally, the simulation thread will signal the results handlers that a computation cycle has been completed and there are results ready.

Continuing Cycle

If instead the process is defined as the continuation of a previous cycle, the process sets the countdown appropriately, then signals the workers to begin their work. The countdown value will be the same as when the new cycle began. If a worker connection has been closed, its corresponding handler will remain active until a new cycle is started. When this occurs, old alert data is used for the remainder of the simulation cycle. Though this does mean that some precomputed values are used for a portion of the cycle, it should not have a significant impact.

5.2.4 Worker Handler

After being notified there is work to be performed, the worker handler can begin its execution. As with the simulation process, the behavior will vary based on the cycle type.

New Cycle

On a new cycle, the handler clears all of its local data. It then takes items off the queue. The exact number is specified by the simulation process. After taking the specified number of entries or emptying the queue, the handler will then gather all of the data the worker client needs to process the pairs of data it will be sent. The handler keeps track of what it has already sent the worker, so it does not resend data. It then notifies the worker client and sends the data. At this point, the handler waits for the client to finish. When the client finishes and returns the results to the handler, the handler places them into a memory space shared with the main simulation process.

If at any point during this process the client signals that it is closing, or heartbeat is not received, signaling a loss of connection, the worker handler places its entries back into the queue to be processed by anther handler. If instead the client does return results successfully, the handler again tries to pull data off of the queue. If the queue is empty, the handler then waits on the countdown object. Unlike the simulation process, this wait has a timeout period. If the countdown reaches zero before the timeout occurs, work has been completed. If it instead times out, there may still be work to do as a connection may have been closed. In this case, the handler again tries to pull more pairs off of the queue so the cycle can be completed.

Continuing Cycle

On a continuation cycle, the worker handler simply signals the worker to begin. It then waits for the work to finish. If the client decides to close, the current countdown is adjusted and data from the previous iterations is passed back to the simulation process. If instead it completes the computations, these results are passed back to the simulation process.

5.3 Implementation

The described system is implemented in C++. While the server and worker clients can operate independently, the configuration, results, and data gathering clients are integrated into the existing CSD system. The architecture of the integrated system can be seen in Figure 5.7. Though the clients are currently integrated, they can be reused with relative ease.

The configuration and data gathering clients are threads that listen for messages. The messages they receive consist of a header, which is an integer specifying the type of data and the actual data. The value used to specify the data is adopted from the CSD implementation. As these clients simply send messages and do not provide any feedback to the creating process, it is not necessary to override any of the contained methods.

The result client is a thread that, upon receiving results, post a message to the thread that created it. As with the other clients, the header specifying the message type is adopted from the CSD system. Future results clients can derive from the base results client and override two methods. The first specifies what, if any, data should be requested after the results have been received. The second is what should occur after the results have been received.

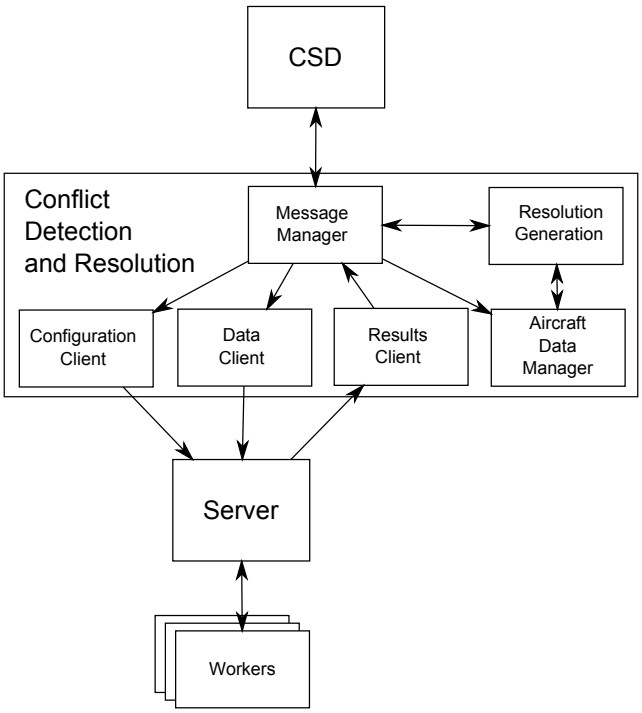

Figure 5.7: Structure of the integrated system

Chapter 6

Evaluation

The testing and subsequent analysis of the system occurred in a number of steps. Prior to integrating into the current CSD system, the underlying architecture was tested. To perform these tests, a number of mock classes were created to simulate various components of the CSD system. The purpose of this testing was to ensure that the design of the underlying architecture functioned as expected with regards to control and data flow prior to being integrated into the CSD system.

After verifying a sound architecture through the mock system testing, work then moved to building the system such that it could be used with the CSD. Due to the restrictions placed on the released code, the testing done was isolated to a single machine. As this was supposed to be a distributed system, this presents a less than ideal situation with regards to testing and evaluation. However, some amount of validation was done to ensure requirements were met.

To test additional clients, a mock data gathering and mock result clients were created. The mock data client request a series of ids to send data on. It timestamps these with a low value so they are not used over actual data. The mock results client similarly requested ids. It then requested results for these aircraft ids from the server. Nothing was done with the data after it was received.

6.1 Clients

The server does not place a limit on the number of client connections that can be made. The performance as the number of clients increases are discussed in Section 6.3.

6.1.1 Data Clients

The system supports two types of data clients, one for aircraft data and another for configuration data. The aircraft data client provided flight plan and state information to the server. This is the minimum amount of information required to create routes for the conflict detection algorithm. Running the simulations required the use of the CSD integrated data clients to perform the analysis.

6.1.2 Worker Clients

The worker clients execute the computational workload of a simulation. The system was run with multiple worker clients. The number of worker clients used ranged from one to sixteen. At sixteen, a limit was reached on the overall processing power of the test machine. As the work done uses the algorithm in use by the FDDRL, the results produced by the system should be similar to the current system. The system also supports performing both active and modified routes simulations simultaneously. Both of these points are further supported in Section 6.3.

6.1.3 Results Clients

The results clients received simulation results after a given cycle. As with the data clients, the integrated results client was used in all the runs used to perform an analysis. The accuracy of the results relied on this client passing information back to the CSD system. Though this is less than ideal, it does provide some information with regards to the accuracy of the results produced by the system.

6.2 Server

The rate at which the server ran was controlled by a configuration value. This value enforced a maximum rate at which a simulation cycle would run. In addition, two unique configurations were communicated to the worker clients. Each configuration was associated with either the active route or modified route simulations.

6.3 Analysis

The accuracy of all of the results produced from the conflict detection algorithm could not be validated. The closest that could be done was a visual comparison of the CSD using the system developed for this project and the CSD using preexisting work done by the FDDRL. Under all of the test that are to be discussed, the results were similar to the current CSD system. Unless otherwise specified, the tests were performed using only active routes.

6.3.1 RAT Usage

The first test done involved using the RAT in both systems. Both systems accurately displayed alerts as the route was modified. However, the results of the developed system took approximately 500ms receive new data, perform the computation, and return the results to the CSD. This is significantly slower than the current system. While modifying a route, the current system provides instant feedback to potential conflicts on the route. With regards to the requirement of integrating and completing simulations in a reasonable amount of time, the system falls short when used with the RAT. This tool requires near instant updates to be useful. However, the 500ms time to complete a computation is acceptable for the active route conflict detection as the current system is configured to produce results once a second.

Number Data Clients	Time to Collect Data (ms)
1	6.59802
2	7.52988
4	8.65189
8	10.5986
16	14.9069
32	28.7138

Table 6.1: Table of average times to collect aircraft data.

6.3.2 Data Gathering Clients

This test involved increasing the number of data clients and analyzing the effect it has on the server. Running these test involved a the CSD system, the server, a single worker, and a variable number of mock data gathering clients. The mock gathering clients provided the same number of state entries as the CSD system. The actual content of these entries is irrelevant as the process always takes the data provided by the CSD system over the mock clients.

The test was run over a 5 minute period while increasing the number of data gather clients connected to the system. The number of data clients attached to the system effects the performance of the data collection task when a new simulation cycle begins. The times generated are based on a time taken before and after the snapshot occurs. The number of clients in this test ranged from 1 to 32, following an 2^n growth pattern. The average times produced from can be seen in Table 6.1 and Figure 6.1. The times have a polynomial growth rate. Though less than ideal, the time used to collect data remained relatively short. In addition, this test provided a complete set of repeat entries. For each client connection, all of the available entries are iterated over. By reducing the number of repeated entries, the time may decrease. In addition, a faster locking mechanism may be utilized to further improve the performance of the data gathering tasks.

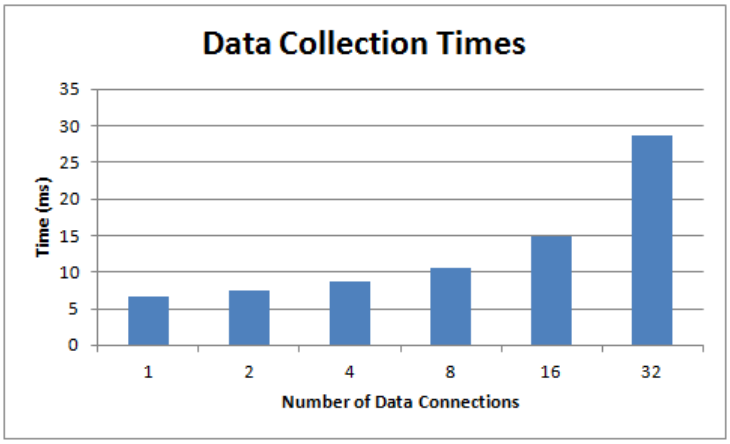

Figure 6.1: Graph of times to collect aircraft data.

6.3.3 Worker Clients and Requested Aircraft

This test involved increasing the number of worker clients and the number of requested aircraft. The worker clients increased from 1 to 16, following an 2^n growth pattern. Testing stopped at 16 because the limit of the processor had been reached. The number of requested aircraft consisted of 1, 10, and 20. The testing was done using the CSD system, a variable number of worker clients and one mock results client. Additional testing was done with an increased number of results clients, but the times gathered were similar to those of a single mock result client. This is expected as all of the results clients access a single object containing alert data using a shared lock. As they are all reading, they can all access the data simultaneously. There is no expected slowdown from additional threads requesting results data at the same time.

Again, the simulations were run over a 5 minute period. The data collected was the time taken to complete the computations, time to gather the results, and time to insert the results.

Number Workers	1	2	4	8	16
1 Request	194.064	160.478	180.426	256.230	353.384
10 Request	869.246	736.8945	862.103	1177.27	2014.36
20 Request	2142.12	1589.87	1741.83	2210.22	3723.87

Table 6.2: Table of average times (in ms) to complete work for variable number of ownship request and worker clients.

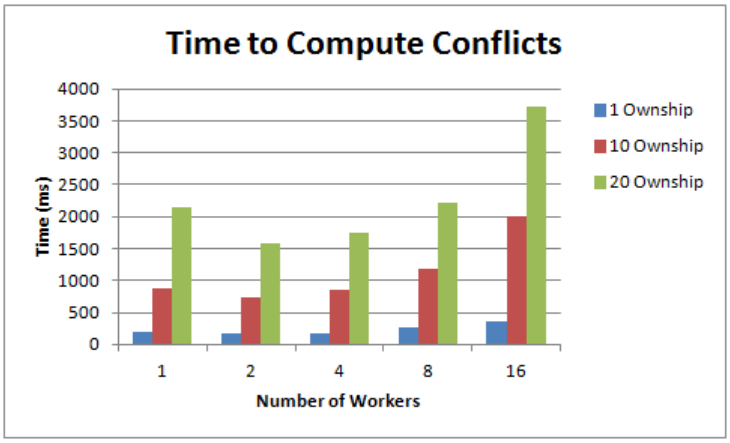

Figure 6.2: Graph of times to compute conflicts.

The time to complete work was measured by capturing a time prior to signaling work was ready, and again after work was completed. The results can be seen in Table 6.2 and Figure 6.2. Regardless of the number of ownship request, the growth pattern stayed the same as the number of workers used increased. Recall that all of these clients were executed on the same machine. Thus the time to complete a given computation will likely decrease as work is moved to other machines. It is interesting to note that with two and four worker clients, the average time to complete the computations decreased when compared to one across all of the ownship requests. The exact cause of this is unknown.

The time to gather results was collected in a similar fashion to the previous times. The average times can be seen in Table 6.3 and Figure 6.3. Notice this also has a polynomial growth rate as seen in the times of the other components.

Number Workers	1	2	4	8	16
1 Request	0.055627	0.212319	0.244094	0.507668	0.690607
10 Request	0.904615	1.17006	1.78214	2.89861	4.82407
20 Request	1.77777	2.32098	3.38787	5.21186	8.94444

Table 6.3: Table of average times (in ms) to collect alert data for variable number of ownship request and worker clients.

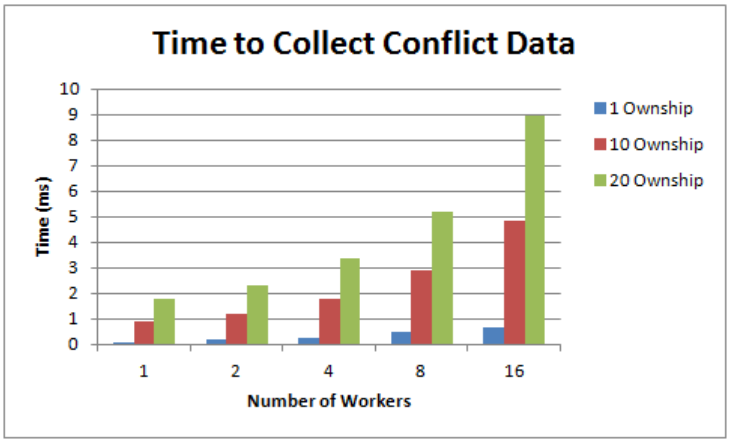

Figure 6.3: Graph of times to collect alerts.

The final times taken were the amount of time to insert the results. However, these returned values of between 0 and 1 millisecond. As such, no further analysis is performed on these values. However, the fact that they remained constant is expected as the number of ownship aircraft requested by a results client should have a minimal effect on the time it takes to insert data into the shared structure.

All of the times gathered displayed a polynomial growth rate as additional clients were added. This is less than ideal since, for a truly scalable application, a slower growth rate is desired. However, the system does provide a number of enhancements over the current CDnR system used by the CSD. Mainly the ability for multiple clients to request multiple aircraft simultaneously.

The purpose for building out a distributed system is to allow computations to be spread across multiple machines. Because of this, it a direct comparison cannot be made with the current CSD system. The closest comparison that could be made is running a single ownship request with the system. However, distributed systems are expected to be slower than a threaded counterpart at smaller numbers. So even this is not a great comparison as the distributed system is expected to be slower due to the additional overhead cost in setting up and transferring data to the relevant processes.

In addition, these times were generated in an unrealistic scenario. By running all of the test on a single machine, poor performance is expected. These test do demonstrate that a number of clients can be simultaneously connected to the system, and the system still operates, albeit slower than desired.

Chapter 7

Future Work

As this is a proof of concept system, there are still several areas that can be improved on. In addition, as it is replacing a component of another system, care must be take while continuing through the development cycle.

7.1 System Enhancements

Though this project provides a basic system, there are still a number of overall system enhancements that can be made. However, some of these will require additional modification to the current CSD and CDnR systems.

One possible enhancement is the moving the resolution component of the system to the distributed framework. Though only briefly discussed, work is being done to automatically define resolved routes. It may be desirable to move this to the distributed service along with the conflict detection. This would allow for automated route proposals to be quickly shared with affected parties, as is the case with conflict information.

Another improvement is a more fined grained approach to configurations of the simulations. As mentioned, the main execution process uses its own defined configuration. Furthermore, though active and modified routes have there own configurations, they are the same across all simulations of the same type. It may be desirable to allow for a configuration to be specified on a per simulation basis, rather than applying a blanket configuration.

As mentioned at the beginning of the paper, this project was designed to be implemented with the current CSD system. Because of this, there are some inherent limitations with providing modified route data and requesting results of multiple clients simultaneously. Though this is okay for the CSD system as there is a one to one mapping between an individual CSD and an aircraft, the ability to share data means that this may not always be true. Though the CSD is only interested in a single aircraft, a new, yet to be developed client may require alert data or provide route data for multiple aircraft simultaneously.

Though the developed system manages failures of clients, it does not manage failures of the master. Having a single server creates a single point of failure. As the conflict detection system plays a critical role in the next generation traffic management systems it is important that this failure point is removed. How this is managed will likely involve modifications in how the flight data is stored. At present, it is not persisted anywhere. Thus, if a failure occurs, all of the flight data is lost. With that in mind, another enhancement is moving the data store components to a separate server. This new server may store the data in some custom way on disk. However, a more likely solution would be to leverage existing database technologies. Databases are designed to manage large amounts of data entry and retrieval. Because there is so much work being done in that area, there is no reason this system cannot leverage it.

7.2 Performance Improvements

As mentioned in Section 6, the overall performance of the system, on a small scale was slower than that of the current system. To help minimize this difference, one area that can be improved is the load balancing. There is no intelligence behind how work is distributed on a worker by worker basis. By using a queue, workers that complete work faster can, theoretically execute more work. However, they would have to be nearly twice as fast as another worker which is unlikely. Because the number of entries to remove is static, a slower worker may take more data than it compare entries than it should for an optimal distribution scenario. Future work can return data with each heartbeat to help describe the load of an individual worker. This can then be used to create a more accurate value for entries to remove. Though this will still likely use the static number as a reference.

In addition, the underlying algorithm used to perform the conflict detection can be updated. It was not designed to perform well in a distributed computing structure. In addition, it does not fully utilize the hardware of the running system. Because each simulation entry is independent of the other, this algorithm lends itself well to multi threading. In addition to performing parallel processing on the CPU [21], this algorithm can also leverage general purpose graphics processors [19, 20].

7.3 Testing

As previously described large scale testing was not performed as part of the proof of concept testing. In lieu of this, further testing will be described here.

Many of the same test done on the single machine also need to be done across multiple machines. This would provide a more accurate guide to the overall performance of the system. In addition, the test provided here focused on the overall performance. The accuracy of the results was simply a visual comparison of the results on the CSD system. This also needs to be validated in a more formal manner.

Chapter 8

Contributions and Conclusions

The manual process currently used to maintain safe routes of aircraft will not scale well as the number of aircraft increases. Through leveraging new GPS tagging and communication systems, more accurate data will be available to both pilots and air traffic controllers. This data will be used to alleviate the workload of air traffic controllers by delegating the detection and resolution of unsafe routes to both automation services and pilots.

This project creates a distributed computational service in support of this delegation of work. This allows for a single service to manage the computations of conflicts rather than each individual aircraft. In addition, with a service performing the computations, non aircraft clients may request information about conflicts. This is particularly useful as air traffic controllers will also want to be aware of potential conflicts of the aircraft in a given region. This will also allow for multiple groups, such as pilots and air traffic controllers to collaboratively define new routes.

The contributions of this project are the development of the distributed conflict detection application. With this, multiple clients can request conflict data from any arbitrary aircraft. In addition, several fixes are made to the existing source code while developing this application. Though this does not provide any sort of break through with regards to distributed computing in general, it instead works to establish services to be used as part of future air traffic control systems.

Bibliography

[1] 3D Cockpit Displays of Traffic Information (CDTI). http://humansystems.arc.nasa.gov/groups/FDDRL/technologies/3dcdti.

[2] Ballin, Mark and Hoekstra, Jacco and Wing, David and Lohr, Gary. NASA Langley and NLR Research of Distributed Air/Ground Traffic Management. In *Aviation Technology, Integration, and Operations (ATIO) Conferences*, pages –. American Institute of Aeronautics and Astronautics, Oct. 2002.

[3] Ballin, Mark G and Sharma, Vivek and Vivona, Robert A and Johnson, Edward J and Ramiscal, Ermin. A flight deck decision support tool for autonomous airborne operations. In *Proceedings of the AIAA Guidance, Navigation and Control Conference, Monterey, CA, USA, AIAA*, 2002.

[4] Canton, Riva and Refai, Mohammad and Johnson, Walter W and Battiste, Vernol. Development and integration of human-centered conflict detection and resolution tools for airborne autonomous operations. In *Proc. 12th International Symposium on Aviation Psychology, Oklahoma City, OK*, 2005.

[5] Conflict Detection and Resolution (CDnR). http://humansystems.arc.nasa.gov/groups/FDDRL/technologies/conflict.

[6] Dao, Arik-Quang V. and Brandt, Summer L. and Bacon, L. Paige and Kraut, Joshua M. and Nguyen, Jimmy and Minakata, Katsumi and Raza, Hamzah and Johnson, Walter W. Conflict resolution automation and pilot situation awareness. In *Proceedings of the 1st international conference on Human interface and the management of information: interacting with information - Volume Part II*, HCII'11, pages 473–482, Berlin, Heidelberg, 2011. Springer-Verlag.

[7] Federal Aviation Administration - TCAS. http://adsb.tc.faa.gov/TCAS.htm.

[8] Freiling, Felix C. and Guerraoui, Rachid and Kuznetsov, Petr. The failure detector abstraction. *ACM Comput. Surv.*, 43(2):9:1–9:40, Feb. 2011.

[9] Johnson, Walter and Bilimoria, Karl and Thomas, Lisa and Lee, Hilda and Battiste, Vernol. Comparison of Pilot and Automation Generated Conflict Resolutions. In *Guidance, Navigation, and Control and Co-located Conferences*, pages –. American Institute of Aeronautics and Astronautics, Aug. 2003.

[10] Johnson, Walter and Ho, Nhut and Battiste, Vernol and Vu, KL and Lachter, Joel and Ligda, Sarah and Dao, Arik and Martin, Patrick. Management of Continuous Descent Approach During Interval Management Operation. In *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, pages 4–D. IEEE, 2010.

[11] Johnson, WW and Battiste, V and Granada, S and Johnson, N and Dao, AQ and Wong, D and Tang, A. A Simulation Evaluation of a Human-Centered Approach to Flight Deck Procedures and Automation for En Route

Free Maneuvering. In *International Symposium on Aviation Psychology, Oklahoma City, OK*, 2005.

[12] Jun Wang and Jian-wen Chen and Yong-liang Wang and Di Zheng. Intelligent Load Balancing Strategies for Complex Distributed Simulation Applications. In *Computational Intelligence and Security, 2009. CIS '09. International Conference on*, volume 2, pages 182 –186, Dec. 2009.

[13] Ligda, Sarah V and Johnson, Nancy and Lachter, Joel and Johnson, Walter W. Pilot Confidence with ATC Automation Using Cockpit Situation Display Tools in a Distributed Traffic Management Environment. In *Human Interface and the Management of Information. Information and Interaction*, pages 816–825. Springer, 2009.

[14] Mehta, M.A. and Jinwala, D.C. Analysis of Significant Components for Designing an Effective Dynamic Load Balancing Algorithm in Distributed Systems. In *Intelligent Systems, Modelling and Simulation (ISMS), 2012 Third International Conference on*, pages 531–536, 2012.

[15] Nandagopal, Malarvizhi and Gokulnath, K. and Uthariaraj, V. Rhymend. Sender initiated decentralized dynamic load balancing for multi cluster computational grid environment. In *Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India*, A2CWiC '10, pages 63:1–63:4, New York, NY, USA, 2010. ACM.

[16] NASA Flight Deck Display Research Lab (FDDRL). http://humansystems.arc.nasa.gov/groups/FDDRL/.

[17] Next Generation Air Transportation System (NextGen). http://www.faa.gov/nextgen/.

[18] Nuaimi, K.A. and Mohamed, N. and Nuaimi, M.A. and Al-Jaroodi, J. A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms. In *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on*, pages 137–142, 2012.

[19] NVIDIA Developer Zone. https://developer.nvidia.com/cuda.

[20] OpenCL. http://www.khronos.org/opencl/.

[21] OpenMP. http://openmp.org/.

[22] Othman, Ossama and Schmidt, Douglas C. Issues in the Design of Adaptive Middleware Load Balancing. In *Proceedings of the ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems*, LCTES '01, pages 205–213, New York, NY, USA, 2001. ACM.

[23] Prevot, Thomas and Callantine, Todd and Lee, Paul and Mercer, Joey and Battiste, Vernol and Palmer, Everett and Smith, Nancy. Co-Operative Air Traffic Management: Concept and Transition. In *Guidance, Navigation, and Control and Co-located Conferences*, pages –. American Institute of Aeronautics and Astronautics, Aug. 2005.

[24] Rahmawan, H. and Gondokaryono, Y.S. The simulation of static load balancing algorithms. In *Electrical Engineering and Informatics, 2009. ICEEI '09. International Conference on*, volume 02, pages 640–645, 2009.

[25] A. Reddy. The Science and Technology of Air Traffic Control. http://arstechnica.com/science/2010/03/the-science-and-technology-of-air-traffic-control/.

[26] Rimal, B.P. and Eunmi Choi and Lumb, I. A Taxonomy and Survey of Cloud Computing Systems. In *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, pages 44–51, 2009.

[27] Senger, L.J. and de Souza, M.A. and Foltran, D.C. Towards a Peer-to-Peer Framework for Parallel and Distributed Computing. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2010 22nd International Symposium on*, pages 127–134, 2010.

[28] Shao, G. and Berman, F. and Wolski, R. Master/Slave Computing on the Grid. In *Heterogeneous Computing Workshop, 2000. (HCW 2000) Proceedings. 9th*, pages 3–16, 2000.

[29] Smith, Nancy and Lee, Paul and Prevot, Thomas and Mercer, J. and Palmer, Everett and Battiste, Vernol and Johnson, Walter. A Human-in-the-Loop Evaluation of Air-Ground Trajectory Negotiation. In *Aviation Technology, Integration, and Operations (ATIO) Conferences*, pages –. American Institute of Aeronautics and Astronautics, Sep. 2004.

[30] Syriani, Eugene and Vangheluwe, Hans and Al Mallah, Amr. Modelling and simulation-based design of a distributed devs simulator. In *Proceedings of the Winter Simulation Conference*, WSC '11, pages 3007–3021. Winter Simulation Conference, 2011.

[31] Thulasidasan, S. and Kasiviswanathan, S. and Eidenbenz, S. and Galli, E. and Mniszewski, S. and Romero, P. Designing Systems for Large-Scale, Discrete-Event Simulations: Experiences with the FastTrans Parallel Microsimulator. In *High Performance Computing (HiPC), 2009 International Conference on*, pages 428 –437, Dec. 2009.

[32] Vu, K.L. and Strybel, T.Z. and Kraut, J. and Bacon, P. and Minakata, K. and Nguyen, J. and Rottermann, A. and Battiste, V. and Johnson, Walter. Pilot and Controller Workload and Situation Awareness with Three Traffic Management Concepts. In *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, pages 4.A.5–1–4.A.5–10, 2010.