JOURNAL OF AEROSPACE INFORMATION SYSTEMS
Vol. 14, No. 9, September 2017

Modeling, Optimization, and Operation of Large-Scale Air Traffic Flow Management on Spark

Jun Chen,* Yi Cao,† and Dengfeng Sun‡
Purdue University, West Lafayette, Indiana 47906

DOI: 10.2514/1.I010533

The nationwide air traffic flow management problem often encounters computational difficulty because it is generally modeled as an integer programming problem that requires computationally expensive optimization algorithms. This paper introduces a customized Spark-based optimization architecture for such large-scale integer programming problems to further speed up the modeling and optimization process, where Spark is a big data cluster-computing platform. First, a novel layered aggregate model is developed for handling flexible rerouting problem, which is not well handled in a previous link transmission model. As an aggregate linear model, the layered aggregate model has the nice features of computational efficiency and scalability, which make it suitable for Apache Spark. By applying a dual decomposition method, the original large-scale problem is decomposed into a number of small subproblems. The optimization proceeds by iteratively solving subproblems and updating Lagrange multipliers. This paper encapsulated the process into the Spark-based data processing model such that the optimization is automatically scheduled to run in parallel. Spark gains efficiency by means of in-memory computing and dynamic schedule allocation. This is demonstrated in the experimental results that are compared to an earlier Hadoop MapReduce-based model, where Hadoop MapReduce is a basic cloud computing framework; the Spark-based model is solved twice as fast as the Hadoop MapReduce-based model.

Nomenclature

$A_{\text{arr}}, A_{\text{dep}}$	=	set of links connecting origin or destination airport
$C_{\text{arr}}(t), C_{\text{dep}}(t)$	=	airport arrival and departure capacity
$C_s(t)$	=	maximum number of aircraft allowed in sector s at time t
D_i^k	=	set of downstream links of link i on layer k
$d^k(\lambda_s(t))$	=	objective of the kth subproblem
$f^k(t)$	=	departures on layer k at time t
$\mathbb{K}$	=	set of layers in simulation and optimization
n^k	=	number of links on layer k
Q_{s_i}	=	set of links inside sector s_i
$q_{im}^k(t)$	=	outflow from link i to link m at time t
$q_{ji}^k(t)$	=	inflow from link j to link i at time t
$\mathbb{S}$	=	set of sectors in the National Airspace System
s_i	=	sector that link i lies in
$\mathbb{T}$	=	planning time horizon of air traffic optimization
T_i^k	=	traversal time of link i on layer k, min
$T_i^k(*)$	=	shortest total traversal time from airport to link i on layer k, min
U_i^k	=	set of upstream links of link i on layer k
$\lambda_s(t)$	=	Lagrange multiplier for sector s at time t

I. Introduction

AIR traffic in the United States is managed by air traffic controllers to ensure safe operations in the National Airspace System (NAS). In periods of high traffic demand, the surge of traffic often leads to congestion. Carefully coordinating flights can effectively alleviate congestion. However, managing flights at the national scale is a challenging task that needs the help of computer-based decision support tools (DSTs). Because of the dynamic nature of air traffic, real-time solutions are critical to the applicability of DSTs. The interval of radar-based position update in the enhanced traffic management system (ETMS) for en route traffic is roughly 1 min. Ideally, a DST should deliver a solution within this time frame. As a result, computational efficiency becomes a concern in air traffic modeling. In air traffic management (ATM) research, integer programming (IP) is a common way to formulate scheduling problems, such as runway scheduling [1,2], arrival sequencing [3–5], and rerouting [6,7]. These discrete-time models, solved by the integer optimization approach, often encounter computational issues. This is because the integer optimization approach often requires developing a search tree for integer solutions. The Bertsimas–Stock-Patterson (BSP) model [8] is among the seminal paradigms that address traffic flow management (TFM) problem in the NAS. A medium BSP instance involving 1000 flights can lead to an IP problem with hundreds of thousands of variables and constraints. Solving IP problems at this scale is computationally demanding due to the problem size. According to [9], 2.5 h was needed to finish such a BSP instance on a Sun SPARC station, which was insufficient for operational implementation.

Received 19 January 2017; revision received 15 May 2017; accepted for publication 9 June 2017; published online 5 July 2017.

*Ph.D. Student, School of Aeronautics and Astronautics; chen1241@purdue.edu.
†Ph.D. School of Aeronautics and Astronautics; garfieldcao@gmail.com. Member AIAA.
‡Associate Professor, School of Aeronautics and Astronautics; dsun@purdue.edu. Senior Member AIAA.

To reduce the running time due to optimization (computational efficiency discussed hereafter is measured by wall-clock time), researchers have made considerable strides. On the modeling side, to overcome the computational limitation of the traditional Lagrangian models (e.g., BSP), the Eulerian model of air traffic flow was proposed [10]. Because this modeling technique spatially aggregates the air traffic, its computational complexity does not depend on the number of aircraft but only on the size of the network problem. Later, an aggregate Eulerian–Lagrangian model was proposed to eliminate the splitting and diffusion problems of some Eulerian models by taking into account the origin–destination information of the flights [11]. Following this, a link transmission model was developed based on the Eulerian–Lagrangian model to further improve the computational efficiency [12]. Moreover, a dual decomposition method was introduced to the traffic flow optimization such that the original problem could be divided into small independent problems to perform parallel computing [13]. In this paper, we will develop, implement, and validate a new network-based modeling method, which is highly adaptable to be customized according to geographical features and/or TFM needs, among others.

In parallel, computational difficulty can also be overcome by hardware [14]. With a more powerful computer, the BSP benchmark increased to involve 3000 flights, whereas the running time was reduced to around 16 min [15]. Following this, a multithreaded programming approach was employed to achieve further speedup [16]. The implementation enforces the CPU to run at full scale, thereby increasing efficiency. But the parallelism was limited to a standalone computer. More recently, an Eulerian–Lagrangian model was solved by massive parallel computing [11,12]. The running time decreased from 2 h to 6 min by splitting the computations on a cluster of 10 Dell workstations [17]. The design made full use of distributed computation resources to increase efficiency. However, it requires extensive programming skills to implement multithreaded programming on a cluster, such as dealing with communication and synchronization issues. To overcome this limitation, a cloud computing framework with Apache Hadoop MapReduce was implemented to reduce the development workload from multithreaded programming [18], where Hadoop MapReduce is a software framework to process large-scale data in parallel on large cluster. With its built-in fault-tolerance capability, the MapReduce framework could be not only efficient but also robust. However, MapReduce is not well suited for iterative optimization because, in each iteration, the data have to be read from the Hadoop Distributed File System (HDFS), and there is a significant cost of starting and finishing a MapReduce job. To further improve the efficiency, this paper will extend the Hadoop-based air traffic flow management from MapReduce framework to Spark, a cluster computing platform suitable for large-scale data processing [19]. Further speedup could be achieved by Spark's ability to run computation in memory. Moreover, the unbalanced workload limitation on MapReduce could be solved by Spark's dynamic schedule allocation feature, and the Spark framework abstracts away MapReduce implementation details to help reduce the difficulty of programming.

This paper designs and customizes a large-scale IP model that can be efficiently computed by a fast cluster architecture as provided by Spark. This paper will be among the very first studies for a Spark-based customized optimization architecture, which solves traditional intractable large-scale IP problems. The major contributions of this paper are listed as following.

A layered aggregate model (LAM) is introduced, which has two key features: 1) LAM is highly adaptable to be customized depending on the objectives of studies, which is more general and powerful than the previous link transmission model (LTM) [12]; 2) LAM's layer structure is very suitable for parallel computing framework. In this paper, a customized LAM is illustrated as an example to demonstrate the advantage for addressing flexible rerouting issue in the NAS-wide TFM problem, which is not well handled in the previous LTM.

The LAM was refactored to run under a Spark framework to show that LAM with decomposable algorithms can be encapsulated in the Spark-based data processing model to perform parallel computing.

The performance of the Spark-based model is compared with the MapReduce-based model to quantify the improvement in computational efficiency.

It is worth emphasizing that a fast computational framework with the proper model is the key to help deliver real-time solutions for air transportation system. In particular, a faster computational platform can solve larger problems in the same time and solve the same problem more often in face of disruptions. Moreover, real-time solutions are critical to the applicability of TFM models due to the dynamic nature of air transportation system. Therefore, a faster computational framework could have significant effect on the improvement of air transportation system.

The rest of this paper is organized as follows. Section II introduces the LAM and the TFM problem. Section III introduces the Spark programming procedure: parameterization of the model based on raw flight data and how the LAM is coded under the Spark framework. Section IV presents simulation results to show the improvement of efficiency. Section V concludes this paper.

II. Layered Aggregate Model and Traffic Flow Management Problem

A. Layered Aggregate Model

The layer is a key component in the LAM, which can be defined based on different airspace structures (e.g., airline, airspace centers, region origin–destination pair, etc.). The benefit is that the dynamics of the air traffic flows on each network layer can be modeled independently but in a same mathematical formulation for all the layers, which is the key feature to perform parallel computing. To achieve high level parallelization in this paper, the NAS network will be duplicated for approximately 4000 layers, corresponding to the number of origin–destination airport pairs (o-d pair) in current operation. As shown in Fig. 1, each layer contains only one o-d pair with a link subnetwork connecting them. The link is an abstraction of passage through a sector, which is a basic airspace session of NAS. The travel time of a link is extracted from historical flight data. A flight path is a sequence of links that connect the departure airport and an arrival airport, and all practical flight paths form a subnetwork for each layer. Notice that a layer does not have to be defined based on o-d pairs. In fact, the concept of layer is highly flexible; it could contain a subnetwork of traffic within an air route traffic control centers, a special used airspace, a collection of airspace of interest, etc., depending on the objectives of studies.

The LAM is developed based on the LTM [12], which has the nice features of computational efficiency and invariable complexity with the number of flights. The LTM establishes a route network based on radar tracks extracted from aircraft situation distributed to industry (ASDI) data compiled by the ETMS [20,21]. However, the LTM does not work well with flexible reroutes when tactical weather avoidance is necessary. These limitations will be enlarged in the Next Generation Air Transportation System when there is 2–3 times of air traffic and most airspace is predicted to be congested [22], while more uncertainties in weather and flight scheduling are expected [23]. However, by defining the layer according to the o-d pair, the LTM can be easily modified into the LAM to handle the rerouting issue in the TFM. The main difference between the LAM and the LTM is that there is a link subnetwork between an o-d pair in the LAM rather than a unique flight path in LTM; therefore, the rerouting in TFM will be handled by LAM.

For each layer in LAM, using the principle of conservation of flow, the dynamics of the traffic flow can be modeled as

$$x_i^k(t+1) = x_i^k(t) - \sum_{m \in D_i^k} q_{im}^k(t) + \sum_{j \in U_i^k} q_{ji}^k(t), \quad \forall \ \ i \in \{1, \ldots, n^k\}, \qquad k \in \mathbb{K}, \qquad t \in \mathbb{T} \tag{1}$$

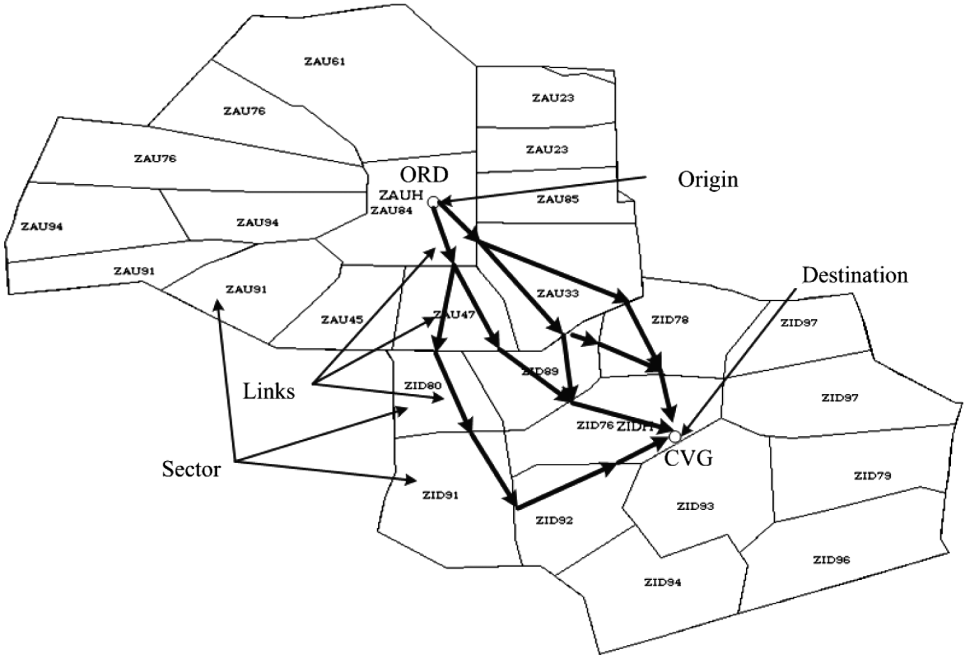


Fig. 1 Conceptual example for subnetwork on ORD-CVG layer.

where the state variable $x_i^k(t)$ is defined as the aircraft count in link i on layer k at time t, $q_{im}^k(t)$ is the outflow from this link i to a downstream link m, and $q_{ji}^k(t)$ is the inflow of link i from a upstream link j. The base model (on one layer) has been validated in numerous previous research such as [22,24]. Notice that the traffic flow model in Eq. (1) is a linear one, which will be a constraint in the optimization problem in Sec. II.B.

B. Traffic Flow Management Problem Based on Layered Aggregate Model

Objective function: The objective is to minimize the weighted total flight time of all flights in the planning time horizon, which reflects the realistic goal to minimize the delay. The objective function can be formulated as

$$\min d = \sum_{t\in\mathbb{T}} \sum_{k\in\mathbb{K}} \sum_{0\le i\le n^k} c_i^k x_i^k(t) \tag{2}$$

Note that c_i^k is the weight for staying in link i on layer k, and $c_i^k = 1$ in this paper; when fairness or equity issues are addressed, c_i^k can assume different values [25,26].

Constraints:

$$\begin{aligned} x_i^k(t+1) &= x_i^k(t) - \sum_{m\in D_i^k} q_{im}^k(t) + \sum_{j\in U_i^k} q_{ji}^k(t) \qquad x_i^k(0) = 0 \\ x_0^k(t+1) &= x_0^k(t) - \sum_{m\in D_0^k} q_{0m}^k(t) + f^k(t), \qquad \sum_{m\in D_i^k} q_{im}^k(t) \le x_i^k(t) \end{aligned} \tag{3}$$

Constraints (3) represent the dynamic of flows as we discussed in Sec. II.A, which enforce conservation of flows. We define a special link 0 ($T_0 = 1$) to represent origin airport; then, its upstream's outflow is the scheduled departure $f^k(t)$, and

$$\sum_{m\in D_i^k} q_{im}^k(t)$$

represents the total outflow of the link i at time t, which should be less than the current state variable $x_i^k(t)$:

$$\sum_{(i,k)\in Q_{s_i}} x_i^k(t) \le C_s(t) \tag{4}$$

$$\sum_{(0,k)\in A_{\text{dep}}} \sum_{m\in D_0^k} q_{0m}^k(t) \le C_{\text{dep}}(t) \tag{5}$$

$$\sum_{(n^k,k)\in A_{\text{arr}}} \sum_{j\in U_{n^k}^k} q_{jn^k}^k(t) \le C_{\text{arr}}(t) \tag{6}$$

Constraints (4–6) enforce en route and airport capacity constraints, where the link n^k is defined as another special link which represents the destination airport in layer k.

$$\sum_{t\in\mathbb{T}}\sum_{m\in D_0^k} q_{0m}^k(t) = \sum_{t\in\mathbb{T}}\sum_{j\in U_{n^k}^k} q_{jn^k}^k(t) = \sum_{t\in\mathbb{T}} f^k(t) \tag{7}$$

Constraints (7) enforce all flights to depart and ensure that all flights land in their destination airport by the end of planning time horizon. Therefore, every scheduled flight job is finished within the planning time horizon to form a complete TFM problem:

$$\sum_{t=T_i^k(*)+T_i^k}^{T_*^k}\sum_{m\in D_i^k} q_{im}^k(t) \le \sum_{t=T_i^k(*)}^{T_*^k-T_i^k}\sum_{j\in U_i^k} q_{ji}^k(t) \tag{8}$$

$$\sum_{t=0}^{T_i^k(*)+T_i^k-1}\sum_{m\in D_i^k} q_{im}^k(t) = 0$$
$$\forall\ T_*^k \ge T_i^k(*) + T_i^k \tag{9}$$

Constraints (8, 9) enforce every flight to stay in a link i for at least T_i^k minutes (the link length), which guarantees that the traffic will not move too fast. The $T_i^k(*)$ represents the shortest total traversal time from the origin airport to link i on layer k:

$$x_i^k(t) \in \mathbb{Z}_+, \qquad q_i^k(t) \in \mathbb{Z}_+$$
$$\forall\ i \in \{1, \ldots, n^k\}, \qquad k \in \mathbb{K}, \qquad t \in \mathbb{T}, \qquad s \in \mathbb{S} \tag{10}$$

Constraints (10) represent integer constraints and define the ranges of the preceding subscripts or superscripts.

Ground delay and airborne delay are two important control strategies in TFM problem, which can be both taken into consideration with LAM. As mentioned in constraints (3), a special link 0 ($T_0 = 1$) is defined to represent the airport as the start of each layer. The control imposed on link 0 corresponds to ground delay. At each time, when an amount of $f^k(t)$ flights enter link 0, the flights are considered to join in a departure queue. Some flights may suffer ground holding at the next time step for traffic congestion. Correspondingly, the controls imposed on the other links are considered to be airborne delay. Note that the weight c_0^k and c_i^k can be adjusted as the costs for ground delay and airborne delay, respectively.

The solution to the preceding TFM problem is the optimal traffic flow as well as the flow control for each layer. Specifically, vector $[x_1^k(t), x_2^k(t), \ldots, x_{n^k}^k(t)]$ represents the state of layer k at time t. As t evolves, the states of the vector represent the movement of traffic flow. Given that real traffic control is generally applied to individual aircraft rather than a flow, the flow control obtained from this model seems impracticable. However, a disaggregation process can convert the flow control into flight-specific actions. The idea is that these optimal states, i.e., vector $[x_1^k(t), x_2^k(t), \ldots, x_{n^k}^k(t)]$, can be used as constraints for scheduling the flights on layer k where variables are defined as ground delays and airborne delays associated with individual flights [12]. The disaggregation process is discussed in detail in [24]. After the disaggregation process, the flow controls are translated into delays imposed on individual flights in each sector.

C. Dual Decomposition Method

The preceding TFM optimization problem has a very large scale. The problem can be seen as overlapping K ($K \approx 4000$) layers of identical network together, with each layer containing network flows from one source (origin airport) to one sink (destination airport). The only complication between these layers is the constraints in Eq. (4) and the departure/arrival capacity constraints where air traffic on multiple layers share a same resource (such as $C_{\text{dep}}(t)$). A dual decomposition method is presented in [13,17] to solve the large-scale linear/integer programming for TFM problems. The decomposition method can be refined for the optimization model in this paper.

By assigning Lagrange multipliers $\lambda_s(t), \lambda_{(0,k)}(t), \lambda_{(n^k,k)}(t)$, these coupled constraints are incorporated into the objective function. Then, the formulation can be rearranged in terms of layer such that each layer is associated with an independent and smaller IP problem. As such, the large-scale problem is decomposed layer by layer into small subproblems. The optimization proceeds by iteratively solving all subproblems and updating Lagrange multipliers based on the solutions of subproblems [13]. The process is summarized in Algorithm 1.

Algorithm 1 Decomposed TFM optimization based on LAM

1: initialization: $j = 0, \lambda_s(t) = \lambda_{(0,k)}(t) = \lambda_{(n^k,k)}(t) = 1$

2: $\forall\ k \in \mathbb{K}$, solve:

$$\min d^k(\lambda_s(t)) = \textstyle\sum_{t\in\mathbb{T}}(\sum_{i=1}^{n^k-1}[c_i^k + \lambda_s(t)]x_i^k(t) + [c_0^k + \lambda_{(0,k)}(t)]x_0^k(t) + [c_{n^k}^k + \lambda_{(n^k,k)}(t)]x_{n^k}^k(t))$$

subject to constraints (3, 7–10)

3: update master problem and Lagrange multipliers:

$$d(\lambda_s(t)) = \max\{\textstyle\sum_{k\in\mathbb{K}} d^k(\lambda_s(t)) - \sum_{t\in\mathbb{T}}\{\sum_{s\in\mathbb{S}}\lambda_s(t)C_s(t) + \sum_{(0,k)\in A_{\text{dep}}}\lambda_{(0,k)}(t)C_{\text{dep}}(t) + \sum_{(n^k,k)\in A_{\text{arr}}}\lambda_{(n^k,k)}(t)C_{\text{arr}}(t)\}\}$$

$$\lambda_s(t) \coloneqq \max\{0, \lambda_s(t) + (1/j+1)(\textstyle\sum_{(i,k)\in Q_{s_i}} x_i^k(t) - C_s(t))\}, \quad \forall\ t \in \mathbb{T}, \quad s \in \mathbb{S}$$

$$\lambda_{(0,k)}(t) \coloneqq \max\{0, \lambda_{(0,k)}(t) + (1/j+1)(\textstyle\sum_{m\in D_0^k} q_{0m}^k(t) - C_{\text{dep}}(t)\}, \quad \forall\ t \in \mathbb{T}, \quad (0,k) \in A_{\text{dep}}$$

$$\lambda_{(n^k,k)}(t) \coloneqq \max\{0, \lambda_{(n^k,k)}(t) + (1/j+1)(\textstyle\sum_{j\in U_{n^k}^k} q_{jn^k}^k(t) - C_{\text{arr}}(t)\}, \quad \forall\ t \in \mathbb{T}, \quad (n^k,k) \in A_{\text{arr}}$$

$$j = j + 1$$

III. Spark Programming Procedure

A. Parameterization of the Traffic Model

To set up the LAM, the following information is needed: 1) layer identified in the NAS and link subnetwork of each layer, 2) the traversal time of each link T_i^k, and 3) scheduled departures $f^k(t)$. All of the information can be obtained by analyzing raw flight data [27]. The ASDI data provide a rich source of historical traffic records for a variety of analysis purposes. However, the flight information is buried in tons of raw flight messages that are recorded in ASDI format. Those messages need be decoded to extract information of interest. An ASDI log recording a full day operations is usually of 500–1000 MB. A snippet of the data looks as follows:

```
...
01F323201952KZHNTZ N717U/041 179 000 3627N/09305W
01F323201952KZHNTZ UAL45/396 520 350 2205N/15422W
02F723201957KZLADZ SWA761/320T/B733/I SAN D2020 ELP 2137
...
```

Each line is a flight message sent by one of the air traffic control facilities across the country. Messages used to restore a flight trajectory include departure message (DZ), arrival message (AZ), flight plan information (FZ), flow control track (TZ), and flight cancellation message (RZ). Because the incoming messages are sorted by timestamp, messages pertaining to the same flight could scatter through an ASDI file. This is typically true for long-range flights. As a result, the whole file needs to be scanned to connect each piece of information associated with a specific flight. Both departures and trajectories are extracted from the data. Departure information is used to generate the time series $f^k(t)$, and trajectories presented as sequences of links are clustered to generate link statistics. In this research, we developed the following procedure to process and analyze the ASDI data to build the mathematical LAM model.

Decoding: ASDI data are highly structured, with each message starting at a timestamp in conjunction with message type, followed by aircraft identification (ACID) and varying number of fields defined for specific message types. Following the definitions of fields specified in the ASDI interface document [20], flight information can be extracted into a list [20]. Information that is necessary for route network construction includes timestamp, message type, ACID, aircraft type, origin and destination airports, geographic coordinates (latitude, longitude, altitude), and speed. An example is shown in Table 1.

Sorting and matching: Grouping the decoded flight messages by ACID yields a flight history of each aircraft. But an aircraft may fly connecting flights during a day. We have to segment connecting flights to extract each individual flight trajectory. This can be done by pairing up neighboring DZ and AZ messages if their origin and destination airports match the ones recorded in the nearest FZ message. Ideally, DZ and AZ messages should appear alternately in an ACID-sorted message sequence [$\boldsymbol{DZ}, TZ\ \cdots, TZ\boldsymbol{AZ}, \cdots, FZ, \boldsymbol{DZ}, TZ\ \cdots, TZ\boldsymbol{AZ}$]. However, the conditions vary as follows.

Incomplete flight records: For unknown reasons, some messages are lost (e.g., [$DZ, \cdots\ DZ, \cdots$]). In this case, we simply discard the DZ or AZ messages that cannot be paired up.

Amended departure or arrival messages: From time to time, multiple departure and arrival messages are filed due to change of flight schedule (e.g., [$DZ, AF, DZ, TZ, \cdots, TZ, AZ, AZ$]). The latest DZ or AZ messages reflect the updates, are thereby retained.

Once DZ and AZ messages are paired up, any message filed in between is associated with the same flight. The ACID-sorted messages in Table 2 show an example.

Aircraft "DAL217" took off from Atlanta International Airport at 00:00:18 and arrived at Tallahassee Regional Airport at 00:37:01.

Filtering: ASDI data are flawed with mismeasurements [28]. Some waypoints deviate from the nominal flight route with large displacements. These anomalies would lead to erroneous link identification in the subsequent step. Therefore, a trajectory needs to be scanned and filtered to eliminate anomalies. The speed and the rate of descent/climb of civil flight are used as the conditions of filtering process.

Table 1 The flight data after decoding

Time	Message	ACID	Aircraft type	Origin	Destination	Latitude	Longitude	Altitude	Speed
00:00:00	FZ	N550DL	C550	KPHN	KEWR	— —	— —	40,000	390
23:59:39	TZ	TRS148	— —	— —	— —	33.01	82.78	39,000	434
23:59:39	TZ	AWE959	— —	— —	— —	34.05	84.91	33,100	457
23:59:39	TZ	N925DW	— —	— —	— —	33.75	85.12	21,000	261
23:59:40	AZ	BTA6044	— —	— —	KORD	— —	— —	— —	— —
23:59:41	TZ	AWE1491	— —	— —	— —	34.79	82.41	12,900	286
23:59:43	AF	N3030	BFM	— —	— —	— —	— —	7,000	160

Table 2 The sorted flight data

Time	Code	ACID	Aircraft type	Origin	Destinatoin	Latitude	Longitude	Altitude	Speed
00:00:18	DZ	DAL217	MD88	KATL	KTLH	— —	— —	— —	— —
00:00:47	TZ	DAL217	— —	— —	— —	33.63	84.48	2400	187
00:01:08	TZ	DAL217	— —	— —	— —	33.6225	84.50	3100	221
00:01:12	TZ	DAL217	— —	— —	— —	33.61	84.51	3500	226
				...					
00:32:33	TZ	DAL217	— —	— —	— —	30.44	84.27	1400	165
00:37:01	AZ	DAL217	— —	KATL	KTLH	— —	— —	— —	— —

Table 3 The flight data after sector mapping

ACID	Time	Origin	Destination	Latitude	Longitude	Altitude	Sector
AWE948	14:19:23	KPHX	KSFO	36.571	−76.342	10,200	ZOA33
AWE948	14:20:23	KPHX	KSFO	36.592	−76.381	10,200	ZOA33
AWE948	14:21:23	KPHX	KSFO	37.008	−76.438	10,200	ZOA32
AWE948	14:22:23	KPHX	KSFO	37.140	−76.479	10,200	ZOA32
AWE948	14:23:23	KPHX	KSFO	37.196	−76.542	10,200	ZOA32
AWE948	14:24:23	KPHX	KSFO	37.244	−76.591	10,000	ZOA32
AWE948	14:25:23	KPHX	KSFO	37.254	−76.642	10,000	ZOA32
AWE948	14:26:23	KPHX	KSFO	37.341	−76.685	10,000	ZOA32
AWE948	14:27:23	KPHX	KSFO	37.406	−76.738	10,000	ZOA31
AWE948	14:28:23	KPHX	KSFO	37.441	−76.794	10,000	ZOA31
AWE948	14:29:15	KPHX	KSFO	37.506	−76.847	8,000	ZOA31
AWE948	14:30:15	KPHX	KSFO	37.597	−76.883	8,000	ZOA31
AWE948	14:31:15	KPHX	KSFO	37.625	−76.916	7,000	ZOA31
AWE948	14:32:15	KPHX	KSFO	37.716	−76.947	6,000	ZOA31
AWE948	14:33:15	KPHX	KSFO	37.811	−76.978	6,000	ZOA31
AWE948	14:34:15	KPHX	KSFO	37.921	−77.017	6,000	ZOA31
AWE948	14:35:15	KPHX	KSFO	38.063	−77.048	5,000	ZOA28
AWE948	14:35:58	KPHX	KSFO	38.147	−77.087	4,000	ZOA28
AWE948	14:36:46	KPHX	KSFO	38.197	−77.102	4,000	ZOA28

Notice that when those "error" messages are discarded, they are also subtracted from the capacity constraints, and the reduction of the capacity is done only for the certain time period. Therefore, removing observations does not introduce biases because capacity estimates are adjusted accordingly.

Interpolation: A flight is tracked by different facilities as it travels along its route; as a result, the intervals between successive TZ messages are not constant, ranging from a few seconds to 4 min. In some cases, the loss of messages results in large intervals. This would cause erroneous measurements in the subsequent link identifications. To guarantee a reasonable spatial resolution, each trajectory is scanned and interpolated if the gap between two successive waypoints is larger than 1 min. Interpolation is linear and based on the three-dimensional position and speed of waypoint where interpolation begins.

Sector mapping: Once filtered, the trajectory is represented by a vector of geographic coordinates, which needs to be mapped into a sequence of sector (defined as polygons in the airspace). Mapping coordinates to sectors can be done by using the ray-casting algorithm [29], which is able to determine whether a point is inside of a polygon. With the geographic coordinates of sectors provided by the Future ATM Concept Evaluation Tool (FACET) [30], the mapping yields Table 3.

Link subnetwork representation and link statistics: The traversal time T_i^k is one of the key parameters for the model setup. For each individual flight, the link traversal time T_i^k can be obtained by calculating the difference of the entry record and exit record to/from a particular sector. However, because individual flight trajectory may temporarily deviate from its scheduled flight plan in reality, the nominal flight route cannot be represented by an individual trajectory. Otherwise, there would be a considerable amount of trivial paths between an o-d pair, creating a huge subnetwork. To construct a reliable link subnetwork representation for each o-d pair, a large amount of trajectories was analyzed, which spans 84 days. The most frequent three routes were used to construct the link subnetwork. For each identified link, the mode of the sample set based on kernel density estimation (KDE) was taken to estimate traversal time of a link [31] because the KDE method provides more accurate prediction compared to traditional method that use the mean [32].

B. Spark Programming

The Apache Spark is a cluster computing platform, which extends the popular MapReduce model to efficiently support more types of computations. One of the main features Spark offers for speed is the ability to run computations in memory, and the system is also more efficient than MapReduce for complex applications running on disk [19]. The resilient distributed data set (RDD) is the basic abstraction in Spark. An RDD in Spark is an immutable, partitioned collection of elements that can be operated on in parallel. In Spark, all work is expressed as either creating new RDDs, transforming existing RDDs, or calling operations on RDDs to compute a result. Spark automatically distributes the data contained in RDDs across a cluster and parallelizes the operations to perform on them. In distributed mode, Spark uses a master/slave architecture with one central coordinator (the driver) and many distributed workers (executors), as shown in Fig. 2. The driver is the process where the main method of the program runs. The driver converts the program into tasks and schedules tasks on executors dynamically based on each executor's

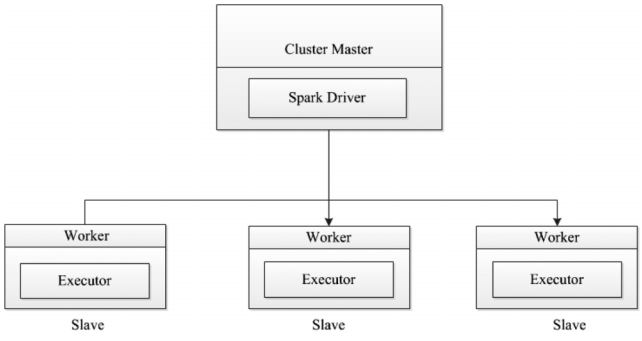


Fig. 2 Distributed Spark system with master/slave architecture.

computational ability, and this dynamic allocation feature is the key to balance workload between workers. After finishing each scheduled task, the worker will return the necessary result to the driver and proceed to the next scheduled task.

In Spark, the RDDs containing key/value pairs are called pair RDDs, which is similar to the concept of the <key, value > pair in MapReduce. Pair RDDs are very useful building blocks for the LAM modeling because they expose operations that allow you to act on each key in parallel or regroup data across the network. Spark's distributed framework and list-processing model make it suitable for the LAM modeling. The LAM problem is developed in two stages. In the first stage, a large ASDI data set is processed to prepare parameters for the LAM setup. Steps described in Sec. III.A are refactored in Spark processes. In the second stage, the recursive optimization algorithm is encapsulated in a chain of Spark transformations and actions. A customized Spark-based optimization architecture for large-scale IP problems is introduced in this stage.

In the first stage, the parameter setup process is similar to the previous work with MapReduce. In MapReduce, the whole process needs to be split and compiled into a job flow, where each job goes through two sequential phases: map and reduce. In map phase, a user-specified function is implemented on all <key, value > pairs in parallel. In reduce phase, pairs with the same key are aggregated with a user-specified rule. In each job, the output of map phase will be the input of reduce phase. After each job, an output file is returned as the input of the downstream job. However, in Spark, the process does not need to follow the standard map and reduce procedure, which reduces the difficulty of programming such that the Spark code is much shorter than MapReduce code. Moreover, all computations can be run in memory such that the output file is unnecessary for chained jobs. A high-level description of the parameter setup process on Spark is provided next. We refer readers to [18] for detailed discussion of the parameters setup process with MapReduce. After the first stage, routes and links (segments of a route in a sector) information is stored in a database for the setup of subsequent LAM optimization.

$RDD_1 = load(ADSIdata)$: read each line of raw flight data and generate RDD.

RDD_1: content of each line of raw messages.

$RDD_2 = map(RDD_1)$: Decoding the flight message and return flight information associated with ACID.

RDD_2: pair RDD <k, v > with k: ACID and v: flight information.

$RDD_3 = reduceByKey(RDD_2)$: Sorting and matching step. Return the flight message associated with each flight.

RDD_3: pair RDD < k, v > with k: ACID + t_{dep} and v: flight information.

$RDD_4 = filter(RDD_3)$: filtering and interpolation step. Check anomaly and interpolate as necessary.

RDD_4: pair RDD <k, v >; with k: ACID + t_{dep} and v: Interpolated $[\varphi(t_m), \lambda(t_m), h(t_m)]$.

$RDD_5 = map(RDD_4)$: sector mapping step. Return sectors associated with ACID + t_{dep}.

RDD_5: pair RDD <k, v > with k: ACID + t_{dep} and v: Sector.

$RDD_6 = reduce(RDD_5)$: link subnetwork representation for each layer and generate its estimated traversal time T_i^k.

RDD_6: pair RDD <k, v > with k: Link name and v: T_i^k.

The Spark process of the LAM optimization starts from the database that we got in the first stage. A flowchart of the Spark process of the LAM optimization is shown in Fig. 3. For each flight, the associated pair RDD is created with the o-d pair (layer) as the key and the departure time as the value. Then, the pair RDDs with the same o-d pair (layer) are aggregated such that each pair RDD is a small subproblem. To set up each subproblem, the departure information $f^k(t)$ is generated, and the traversal time for each link in this subproblem is retrieved from the database. Giving the initial Lagrangian multipliers and the sector capacity information, the subproblem can be solved in parallel. This will be the most computationally intensive step, which solves a large number of IP subproblems in parallel. By aggregating the returned result from all subproblems, the Lagrangian multipliers can be updated, and the algorithm proceeds to next iteration. The workflow is given as follows.

$RDD_1 = load(FlightPlan)$: read each line of flight Plan file and generate RDD.

RDD_1: content of each line of flight plan information.

$RDD_2 = filter(RDD_1)$: filter the flight plan based on selected date and time period.

RDD_2: flight plan on selected date and time period.

$RDD_3 = map(RDD_2)$: create paired RDD with layer name and its departure time.

RDD_3: pair RDD <k, v > with k: layer k and v: departure time t_{dep}.

$RDD_4 = groupByKey(RDD_3)$: group the departure times with the same layer name.

RDD_4: pair RDD <k, v > with k: layer k and v: departure time set $[t_{dep}^1, t_{dep}^2, \cdots]$.

$RDD_5 = map(RDD_4)$: generate departure schedule $f^k(t)$ and retrieve traversal time T_i^k for each link in this layer from database.

RDD_5: pair RDD <k, v > with k: layer k and v: $f^k(t)$ and T_i^k.

$RDD_6 = map(RDD_5)$: set up subproblems with initial Lagrangian multipliers $\lambda_{s_i}(t)$ and the sector capacity information; solve the subproblems concurrently. Return sector name and the sector count contribute by this route.

RDD_6: pair RDD <k, v > with k: sector s_i and v: $x_i^k(t)$.

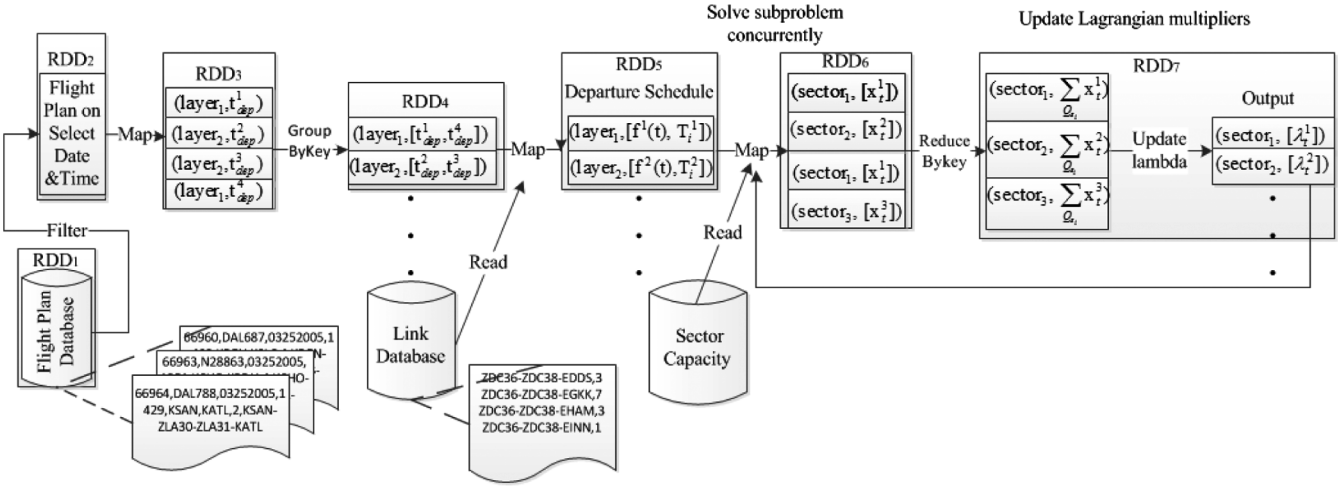


Fig. 3 Flowchart of the recursive algorithm based on Spark.

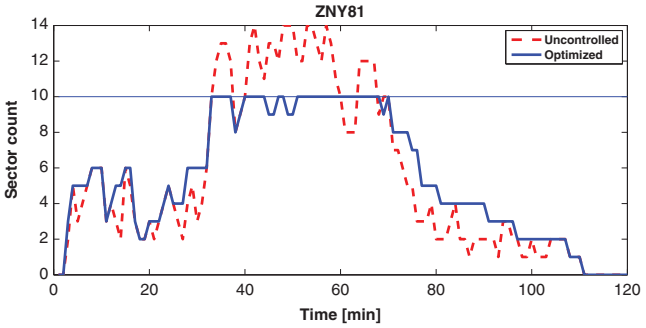


Fig. 4 Comparison of air traffic flow optimization within a sector of airspace.

This will be the most computationally intensive step in the process, which solves a large number of IP subproblems in parallel. The map function here is a customized IP optimization algorithm.

$RDD_7 = reduceByKey(RDD_6)$: aggregate the sector counts contributed by different layers and update Lagrangian multipliers for next iteration.

RDD_7: pair RDD $<k, v>$ with k: sector s_i and v: $\lambda_{s_i}(t)$.

IV. Simulation Results

A. Model Validation

Compared to the LTM, the key advantage of the LAM is the ability to reroute aircraft in TFM. This subsection first validates the optimization algorithm and program. Then, the benefit of the LAM is demonstrated by comparing the LAM's result to the baseline (LTM).

In this experiment, each instance covers traffic in the continental United States airspace in a 2 h period. To get parameters for the LAM, 84 ASDI data files corresponding to the traffic of 84 days were analyzed. As a prototype, this paper only focused on the airspace in the continental United States, which contains 1258 sectors. On each day, over 62,000 flights were tracked in the NAS. Almost 30,000 links and 100,000 routes were identified; 3838 general public airports were included in the route network, which covers the majority of commercial aviation routes in the U.S. airspace. Despite a large volume of data used to provide a statistically significant sample pool, only those involved in the planning horizon were pulled out of the database for the TFM problem setup.

The TFM problem was validated by running a 2 h NAS-wide instance, which represents the high traffic period of a day with 2326 layers and 3054 flights are involved. Given that ASDI traffic is under control in reality, only 70% of the nominal sector capacity is used to test the model by creating a "busy" traffic for all the instances in this paper. The optimized traffic of a representative sector (ZNY81, New York Center) is shown in Fig. 4. It is clear that the optimization alleviates congestion. The sector counts are kept below the sector capacity after the optimization.

To demonstrate the LAM's ability to reroute aircraft in TFM, the LTM's result is set as the baseline and compared with the LAM's result. Without loss of generality, there are 10 2 h NAS-wide instances been used with both the LTM and LAM. Each instance covers a 2 h high traffic period with about 3000 flights and 2000 layers (each layer includes three alternative routes with about 25 links in average). Therefore, each instance has about $2 \times 120 \times 2000 \times 25 = 12{,}000{,}000$ variables and about $120 \times 2000 \times (25+2) = 6{,}480{,}000$ constraints. Table 4 shows the details and statistics. The last column is the computing time ratio based on a standalone computer. It is clear that the LAM reduces both the ground delay and air delay cost by introducing rerouting. However, the LAM is 1.5 times slower than the LTM. The reason is that the LAM constructs a link subnetwork between each o-d pair rather than a unique flight path in LTM, which introduces more variables for each o-d pair (layer). This computational issue could be alleviated because the Spark-based platform could improve the computational performance, which will be demonstrated in Sec. IV.B.

B. Performance Improvement with Spark Platform

This section presents NAS-wide instances of TFM problems that run on a Spark cluster. The goal is to compare the performance of the TFM problems on Spark with that on the existing Hadoop MapReduce framework [18]. To make the comparison meaningful, the simulation experiment was set up with the same hardware parameters. The Spark cluster was launched with six nodes, where each node was Dell workstations configured with an Intel i7 CPU and a 16 GB RAM. All workstations run Ubuntu 14.04 with Spark 1.3.1. The optimization subproblems were solved by calling Gurobi 6.0.2 [33].

Table 4 Comparison of LTM and LAM

Instance	Flights	Sectors (airports)	Routes	Ground cost ratio (LAM/LTM)	Air cost ratio (LAM/LTM)	Computing time ratio (LAM/LTM)
1	3054	93 (43)	5326	0.903	0.823	1.535
2	3272	104 (45)	6132	0.915	0.813	1.462
3	2848	87 (41)	5103	0.905	0.837	1.586
4	3558	98 (44)	5718	0.901	0.783	1.638
5	3011	91 (40)	4730	0.921	0.811	1.473
6	2629	78 (35)	3578	0.890	0.833	1.583
7	2724	86 (38)	4212	0.920	0.811	1.565
8	3127	101 (43)	5210	0.886	0.809	1.498
9	3257	96 (40)	4698	0.901	0.796	1.607
10	2976	95 (39)	4345	0.910	0.812	1.499
Average	3046	93 (41)	4905	0.905	0.813	1.545

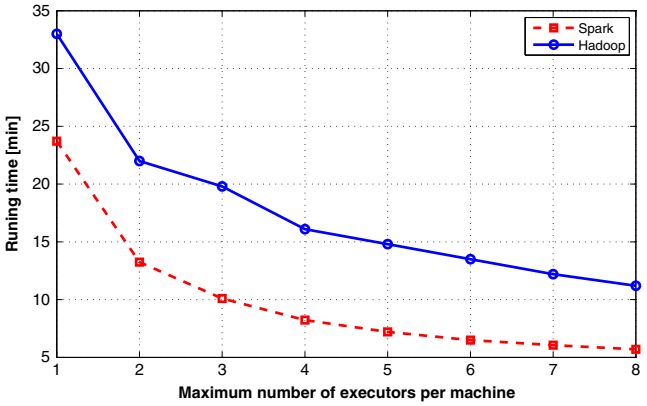
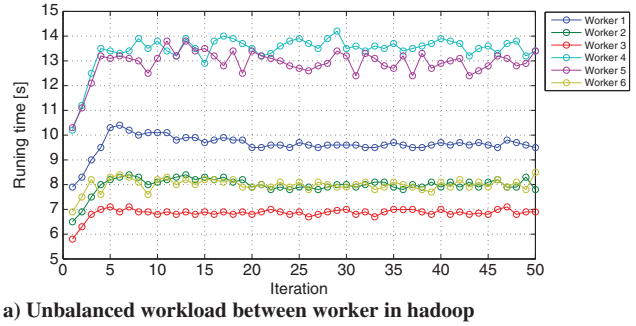


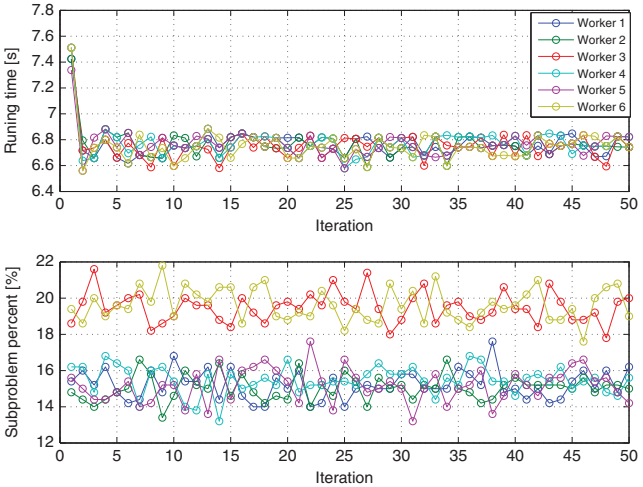
Fig. 5 Running time decreasing as a function of the number of threads per machines.

The performance of the TFM problem on Spark and Hadoop was compared by running the same preceding 2 h NAS-wide instance . The running time of the optimization on Spark and Hadoop is shown in Fig. 5. As a tuning parameter to control the concurrency level, the maximum number of executors allowed on a worker can be set by users. Because the Intel i7 CPU has four physical cores, with each being capable of handling two threads simultaneously, the maximum number of executors per machine can be up to eight. As a result, the maximum number of threads a machine can run in parallel is eight. In theory, the Spark cluster can have at maximum $6 \times 8 = 48$ threads run concurrently. The running time decreases as more executors are launched. However, the speedup is not linear to the number of threads. The reason is that the parallel programming model has inherent overheads such as communication and synchronization between workers.

Comparing the running time of Spark and Hadoop in Fig. 5, Spark is about two times faster than Hadoop. The running time with maximum computing power (eight executors per machine) is reduced from 11.2 to 5.7 min. One of the key reasons is that Spark's in-memory computation cuts down internal input and output processes for iterative jobs. In MapReduce, the input/output data have to be read from/stored to HDFS in each iteration, and there is significant cost of starting and finishing a MapReduce job. However, Spark's in-memory computation avoids such cost that parameter updates can be cached in memory between iterations in the optimization process, which contributes to the speedup. Another key reason is that unbalanced workloads cause idle time for some workers on Hadoop cluster. Figure 6a shows the running time of solving subproblems on each worker of a Hadoop cluster. Worker 3 is about 6.5 s ahead of worker 4 in each iteration. In the implementation of Hadoop, the subproblems were evenly distributed to each worker in the beginning. Although all the workers have the same configuration, the complexity of subproblems has



a) Unbalanced workload between worker in hadoop



b) Runtime and workload between workers in Spark

Fig. 6 Comparison of Hadoop and spark runtime.

Table 5 Comparison of MapReduce framework and spark framework

Features	MapReduce	Spark
Code, lines	942	221
Speedup	15.8	19.9
Workload balancing	No	Yes
Fault tolerance	Yes	Yes

a wide range due to the difference of subnetwork. However, Spark can dynamically allocate subproblems to each of the workers, which helps to reduce the gap. Figure 6b shows the running time and workload of solving subproblems on workers in Spark. Worker 3 and 6 were more powerful than others such that the Spark driver distributed 5% more subproblems to them in the same time. As a result, the runtime is almost the same on each worker in each iteration. This feature of Spark helps avoid the idle time associated with Hadoop cluster such that the average solving time in each iteration is improved.

The main differences between the MapReduce framework and Spark framework are summarized in Table 5. By taking the advantage of Spark's RDD framework, the process does not need to follow the standard map and reduce procedures in the MapReduce framework such that the list-processing job is easier to program with fewer lines of code. Beyond the list-processing job, Spark's RDD framework abstracts away MapReduce implementation details such that it can cover a wide range of workloads to become a capable platform for other large-scale dynamical systems, which is not tractable on a traditional computational platform. In addition, the speedup result shown in Table 5 is compared with the results from a standalone computer.

As a cloud-based computational framework, another advantage of Spark is its built-in fault tolerance capability. A test is shown in Fig. 7 where two workers (workers 3 and 6) were purposely shut down during the iterations. The optimization was held up by the shut down. The master retried to schedule the failure tasks to these two workers for several times. When the master detects that these workers fail to respond for a preset period of time, it reschedules the tasks to be reexecuted on other workers so that the job can continue. In the remaining iterations, Spark recalculated the tasks splits and assigned tasks to the alive workers. As a result, the whole optimization was completed without interruptions. Fault tolerance is an important feature that the non-cloud-based computational framework does not provide.

C. Application

To demonstrate that our model with Spark-based computing framework could help the improvement of air transportation system, we implemented our model and computing framework with realistic traffic data on NASA's simulation platform: the Future ATM Concept Evaluation Tool (FACET) [30]. FACET is a flexible software tool, which provides researchers with a simulation environment for preliminary testing of advanced ATM concepts. In addition to modeling the airspace system for research, FACET has also successfully transitioned into a valuable tool for operational use. Federal Aviation Administration traffic flow managers and commercial airline dispatchers have used FACET for real-time operations planning with live air traffic data.

Figure 8 demonstrates the application framework with live air traffic data. First, a flight plan is fed to the spark-based data process module, where the flight plan is filed by commercial carriers to the air traffic authority approximately 3 h before departure. Second, the spark-based LAM optimization module will solve for the optimal flow schedule with the current flight plan, which only takes several minutes with the distributed Spark-based computing framework. Finally, the optimized result can be validated visually by feeding the optimized result into FACET through its application program interface. Figure 9 demonstrates the visual validation with FACET, which compared the uncontrolled traffic with optimized traffic for the same time step. It is clear that the LAM optimization alleviates congestion. Based on the validation, the traffic flow managers can

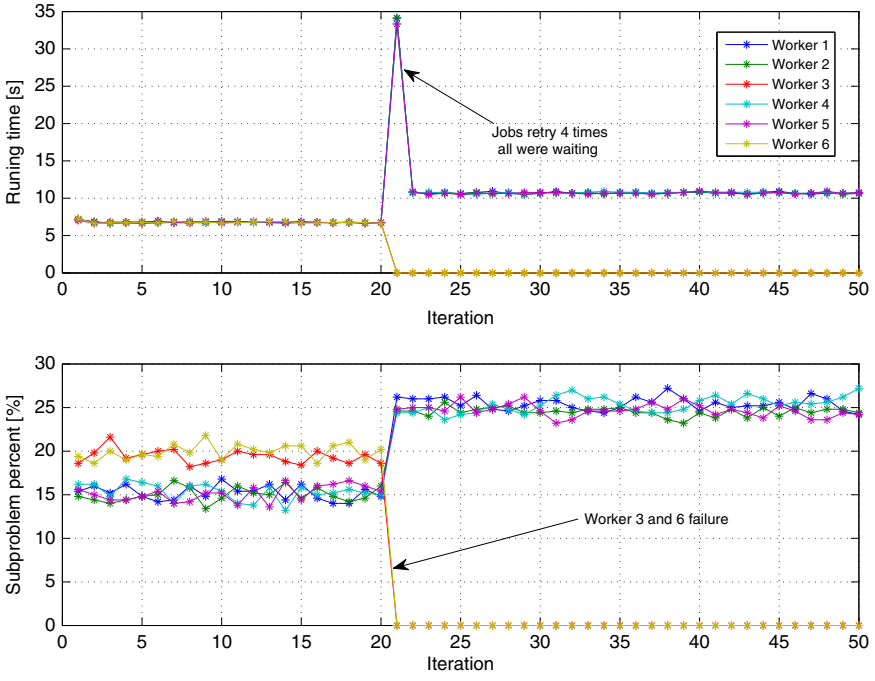


Fig. 7 Failure tolerance test when workers 3 and 6 were shut down.

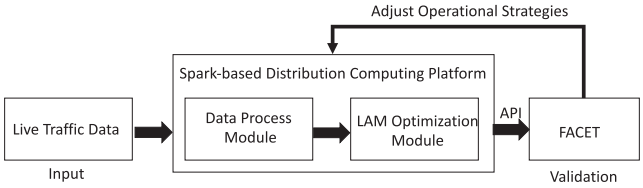
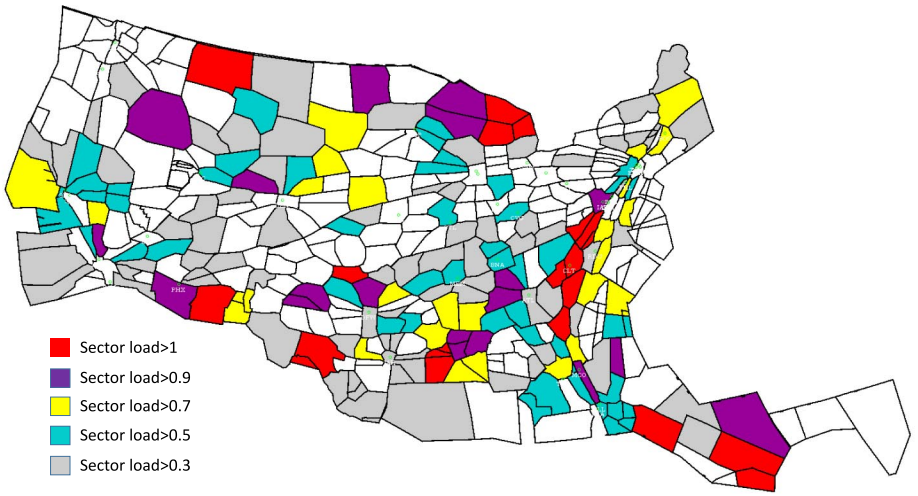
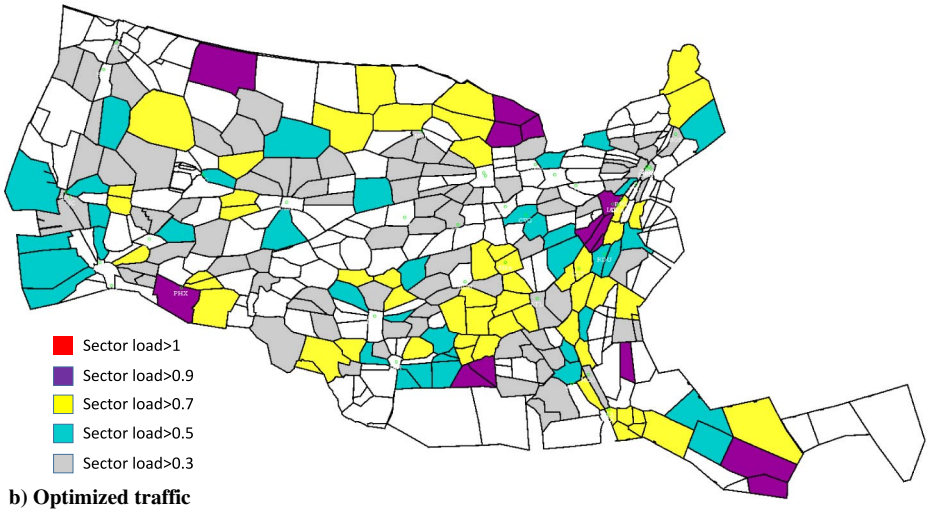


Fig. 8 Application framework with live traffic data.



a) Uncontrolled traffic



b) Optimized traffic

Fig. 9 Traffic simulation in FACET: uncontrolled traffic vs optimized traffic at same time step.

easily adjust their operational strategies. More importantly, the faster computational platform with Spark can provide multiple validation opportunities in the face of disruptions to provide a timely solution, which could have a significant effect on the improvement of air transportation system.

V. Conclusions

This paper introduces a novel layered aggregate model, which is highly adaptable to be customized, depending on different objectives such that it is more general. To demonstrate the advantage over the previous link transmission model, a customized o-d pairs layered aggregate model (LAM) is illustrated as an example, which can address flexible rerouting issue in the National Airspace System-wide traffic flow management (TFM) problem. The large TFM problems based on LAM can be decomposed into smaller subproblems by using Lagrangian methods and employing the parallel programming model. Notice that a layer does not have to be defined based on o-d pairs. In fact, the concept of a layer is highly flexible. One could separately define metroplexes or certain special-use regions of airspace; one might even be able to modify the model to accommodate dynamically changing airspace structure; one might also think of segregating vertically, to model separately unmanned aerial vehicle operations in low-altitude airspace.

Besides, this paper extends the Hadoop-based air traffic flow management problem with MapReduce framework to Spark, a cluster computing platform suitable for large-scale data processing. A customized Spark-based optimization architecture for large-scale integer programming problems is first proposed and tested. In comparison with the MapReduce framework, traffic flow management problems can be solved more

efficiently in Spark. Spark's ability to run computation in memory saves the unnecessary step of generating output file for each job on MapReduce. In addition, the unbalanced workload limitation on MapReduce framework was overcome by Spark's dynamical schedule allocation feature. As a result, further speedup was achieved. Besides efficiency, the Spark framework abstracts away MapReduce implementation details to help reduce the difficulty of programming (as measured by fewer lines of code), and Spark's distributed framework also demonstrates runtime fault tolerance. These features, as demonstrated by our experiments, make the Spark a capable platform that can potentially solve and analyze some large-scale dynamical systems, which is not tractable on a traditional computational platform.

Acknowledgment

The authors are grateful to Kapil Sheth (NASA Ames Research Center) for his help with the Future ATM Concept Evaluation Tool.

References

[1] Balakrishnan, H., and Chandran, B. G., "Algorithms for Scheduling Runway Operations Under Constrained Position Shifting," *Operations Research*, Vol. 58, No. 6, 2010, pp. 1650–1665. doi:10.1287/opre.1100.0869

[2] Wesonga, R., "Airport Utility Stochastic Optimization Models for Air Traffic Flow Management," *European Journal of Operational Research*, Vol. 242, No. 3, 2015, pp. 999–1007. doi:10.1016/j.ejor.2014.10.042

[3] Eun, Y., Hwang, I., and Bang, H., "Optimal Arrival Flight Sequencing and Scheduling Using Discrete Airborne Delays," *IEEE Transactions on Intelligent Transportation Systems*, Vol. 11, No. 2, 2010, pp. 359–373. doi:10.1109/TITS.2010.2044791

[4] Chen, J., DeLaurentis, D., and Sun, D., "Dynamic Stochastic Model for Converging Inbound Air Traffic," *Journal of Guidance, Control, and Dynamics*, Vol. 39, No. 10, 2016, pp. 2273–2283.

[5] Corolli, L., "Deterministic and Stochastic Optimization for Heterogeneous Decision Phases in the Air Traffic Domain," Ph.D. Dissertation, Università degli Studi di Milano-Bicocca, Milan, Italy, 2014.

[6] Agustin, A., Alonso-Ayuso, A., Escudero, L. F., and Pizarro, C., "On Air Traffic Flow Management with Rerouting. Part 1: Deterministic Case," *European Journal of Operational Research*, Vol. 219, No. 1, 2012, pp. 156–166. doi:10.1016/j.ejor.2011.12.021

[7] Agustin, A., Alonso-Ayuso, A., Escudero, L. F., and Pizarro, C., "On Air Traffic Flow Management with Rerouting. Part 2: Stochastic Case," *European Journal of Operational Research*, Vol. 219, No. 1, 2012, pp. 167–177. doi:10.1016/j.ejor.2011.12.032

[8] Vranas, P. B., Bertsimas, D. J., and Odoni, A. R., "The Multi-Airport Ground-Holding Problem in Air Traffic Control," *Operations Research*, Vol. 42, No. 2, 1994, pp. 249–261. doi:10.1287/opre.42.2.249

[9] Bertsimas, D., and Patterson, S. S., "The Air Traffic Flow Management Problem with Enroute Capacities," *Operations Research*, Vol. 46, No. 3, 1998, pp. 406–422. doi:10.1287/opre.46.3.406

[10] Menon, P. K., Sweriduk, G. D., and Bilimoria, K. D., "New Approach for Modeling, Analysis, and Control of Air Traffic Flow," *Journal of Guidance, Control, and Dynamics*, Vol. 27, No. 5, 2004, pp. 737–744. doi:10.2514/1.2556

[11] Sun, D., and Bayen, A. M., "Multicommodity Eulerian–Lagrangian Large-Capacity Cell Transmission Model for En Route Traffic," *Journal of Guidance, Control, and Dynamics*, Vol. 31, No. 3, 2008, pp. 616–628. doi:10.2514/1.31717

[12] Cao, Y., and Sun, D., "Link Transmission Model for Air Traffic Flow Management," *Journal of Guidance, Control, and Dynamics*, Vol. 34, No. 5, 2011, pp. 1342–1351. doi:10.2514/1.51495

[13] Sun, D., Clinet, A., and Bayen, A. M., "A Dual Decomposition Method for Sector Capacity Constrained Traffic Flow Optimization," *Transportation Research Part B: Methodological*, Vol. 45, No. 6, 2011, pp. 880–902. doi:10.1016/j.trb.2011.03.004

[14] Li, W., Dib, M. V. P., Alves, D. P., and Crespo, A. M. F., "Intelligent Computing Methods in Air Traffic Flow Management," *Transportation Research Part C: Emerging Technologies*, Vol. 18, No. 5, 2010, pp. 770–780. doi:10.1016/j.trc.2009.05.015

[15] Bertsimas, D., Lulli, G., and Odoni, A., "The Air Traffic Flow Management Problem: An Integer Optimization Approach," *Integer Programming and Combinatorial Optimization*, Springer, Berlin, 2008, pp. 34–46, https://link.springer.com/chapter/10.1007/978-3-540-68891-4_3.

[16] Rios, J., and Ross, K., "Massively Parallel Dantzig-Wolfe Decomposition Applied to Traffic Flow Scheduling," *Journal of Aerospace Computing, Information, and Communication*, Vol. 7, No. 1, 2010, pp. 32–45. doi:10.2514/1.45606

[17] Cao, Y., and Sun, D., "A Parallel Computing Framework for Large-Scale Air Traffic Flow Optimization," *IEEE Transactions on Intelligent Transportation Systems*, Vol. 13, No. 4, 2012, pp. 1855–1864. doi:10.1109/TITS.2012.2205145

[18] Cao, Y., and Sun, D., "Migrating Large-Scale Air Traffic Modeling to the Cloud," *Journal of Aerospace Information Systems*, Vol. 12, No. 2, 2015, pp. 257–266. doi:10.2514/1.I010150

[19] Karau, H., Konwinski, A., Wendell, P., and Zaharia, M., *Learning Spark: Lightning-Fast Big Data Analysis*, O'Reilly Media, Sebastopol, CA, 2015, pp. 3–6.

[20] Division, V. C. A. A., "Aircraft Situation Display to Industry: Functional Description and Interface Control Document," Volpe Center TR ASDI-FD-001, Cambridge, MA, 2000.

[21] Volpe, N. T. C., "Enhanced Traffic Management System (ETMS)," U.S. Dept. of Transportation, TR VNTSC-DTS56-TMS-002, Cambridge, MA, 2000.

[22] Sridhar, B., Grabbe, S. R., and Mukherjee, A., "Modeling and Optimization in Traffic Flow Management," *Proceedings of the IEEE*, Vol. 96, No. 12, 2008, pp. 2060–2080.

[23] Buxi, G., and Hansen, M., "Generating Day-of-Operation Probabilistic Capacity Scenarios from Weather Forecasts," *Transportation Research Part C: Emerging Technologies*, Vol. 33, Aug. 2013, pp. 153–166. doi:10.1016/j.trc.2012.12.006

[24] Sun, D., Sridhar, B., and Grabbe, S. R., "Disaggregation Method for an Aggregate Traffic Flow Management Model," *Journal of Guidance, Control, and Dynamics*, Vol. 33, No. 3, 2010, pp. 666–676. doi:10.2514/1.47469

[25] Bloem, M., and Sridhar, B., "Optimally and Equitably Distributing Delays with the Aggregate Flow Model," *Proceedings of the IEEE/AIAA 27th Digital Avionics Systems Conference*, IEEE Publ., Piscataway, NJ, 2008, pp. 3–11.

[26] Glover, C. N., and Ball, M. O., "Stochastic Optimization Models for Ground Delay Program Planning with Equity–Efficiency Tradeoffs," *Transportation Research Part C: Emerging Technologies*, Vol. 33, Aug. 2013, pp. 196–202. doi:10.1016/j.trc.2011.11.013
[27] Klein, A., and Hafner, F., "Computation and Analysis of Aircraft Proximities in the NAS Using ETMS Data," *Proceedings of the 24th Digital Avionics Systems Conference, 2005*, Vol. 1, IEEE Publ., Piscataway, NJ, 2005, p. 3-B.
[28] Salaun, E., Gariel, M., Vela, A. E., Feron, E., and Clarke, J.-P. B., "Airspace Complexity Estimations Based on Data-Driven Flow Modeling," *AIAA Guidance, Navigation and Control Conference*, AIAA Paper 2010-8074, Aug. 2010.
[29] Horvat, D., and Zalik, B., "Ray-Casting Point-in-Polyhedron Test," *Proceedings of the CESCG 2012: The 16th Central European Seminar on Computer Graphics*, The Vienna Univ. of Technology, Smolenice, Slovakia, April–May 2012.
[30] Bilmoria, K. D., Sridhar, B., Chatterji, G. B., Sheth, K. S., and Grabbe, S. R., "FACET: Future ATM Concepts Evaluation Tool," *Air Traffic Control Quarterly*, Vol. 9, No. 1, 2001, pp. 1–20.
[31] Silverman, B. W., *Density Estimation for Statistics and Data Analysis*, Vol. 26, CRC Press, Boca Raton, FL, 1986, pp. 34–50.
[32] Cao, Y., "Cloud-Based Large-Scale Air Traffic Flow Optimization," Ph.D. Thesis, Purdue Univ., West Lafayette, IN, 2013.
[33] "Gurobi Optimizer Reference Manual," Gurobi Optimization, Inc., Hounston, TX, 2017, http://www.gurobi.com/documentation/7.0/refman.pdf [retrieved March 2014].

H. Balakrishnan
Associate Editor