Migrating Large-Scale Air Traffic Modeling to the Cloud

Yi Cao* and Dengfeng Sun†
Purdue University, West Lafayette, Indiana 47906-2045

DOI: 10.2514/1.I010150

Coordinating nationwide air traffic flow is a large-scale problem. The modeling process generally involves analysis of massive flight data, and its optimization involves computationally expensive algorithms. This paper uses Hadoop MapReduce, a big data processing model, to facilitate air traffic flow modeling and optimization, where computationally intensive tasks are automatically spread to Hadoop clusters for concurrent executions. The overall wall-clock time of computation is reduced. A nationwide traffic flow management problem that has been previously studied was restructured under the MapReduce framework. The problem aims at minimizing flight delays while respecting system capacities. Due to its temporal and spatial scope, the size of this problem grows to an extent where it is too big to be solved on standalone computers. Lagrangian relaxation was applied to decompose the original problem into a collection of solvable subproblems. The optimization proceeds in two iterative stages: solving subproblems and Lagrange multiplier updates. These two processes are encapsulated in the mapper and reducer functions, respectively. As a result, the optimization is automatically scheduled to run in parallel tasks. The cloud-based air traffic modeling and optimization were validated through running nationwide air traffic optimization instances on a small Hadoop cluster with six nodes. The modeling processing is eight times faster and the optimization is 16 times faster than that running on standalone computers.

Nomenclature

$A_{\mathrm{arr}}, A_{\mathrm{dep}}$	=	set of links connecting origin or destination airport
$C_s(t)$	=	maximum number of aircraft allowed in sector s at time t
$C_{\mathrm{arr}}(t)$, $C_{\mathrm{dep}}(t)$	=	airport arrival and departure capacity
$d^k(\lambda_s(t))$	=	objective of the kth subproblem
$f^k(t)$	=	departures into route k at time t
$\mathbb{K}$	=	set of routes in simulation and optimization
n^k	=	number of links on route k
Q_{s_i}	=	set of links inside sector s_i
$q_i^k(t)$	=	outflow of 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 route k, min
t	=	time step
$\lambda_s(t)$	=	Lagrange multiplier for sector s at time t

Subscripts

i	=	index of a link
j	=	index of iteration
s	=	index of a sector

Superscript

k	=	index of a route

I. Introduction

IN AIR traffic management (ATM), most scheduling problems, such as runway scheduling [1], arrival sequencing [2], and rerouting, are modeled as discrete-time systems and solved by integer programming (IP). Integer optimization is computationally expensive to solve. On the other hand, air traffic control requires real-time solutions to support decision making. As a result, computational issues become a concern in air traffic modeling. This is typically true in strategic traffic management. En route traffic is generally modeled as a multicommodity network. Paradigms involve a queuing network model [3], a Bertsimas–Stock-Paterson (BSP) model [4], and an aggregate flow model [5]. Despite the variations in network abstraction, the underlying formulations are similar, characterized by linear objective functions and integral decision variables. The queuing network model builds a network with tens of thousands of arcs across the National Airspace System (NAS). This model has to restrict route options to mitigate computational issues. The BSP model is designed to reduce NAS-wide delays. Even a medium BSP instance involving 1000 flights can lead to hundreds of thousands of variables and constraints. Calculating integer solutions at this scale is computationally demanding. It is reported that a BSP instance took 2.5 h to finish on a Sun SPARCstation [6]. The interval of radar-based position update in the

Received 28 June 2013; revision received 15 June 2014; accepted for publication 21 December 2014; published online 27 February 2015.

*Ph.D. Student, School of Aeronautics and Astronautics; cao20@purdue.edu. Student Member AIAA.

†Assistant Professor, School of Aeronautics and Astronautics; dsun@purdue.edu. Member AIAA.

enhanced traffic management system (ETMS) for en route traffic is roughly 1 min. Ideally, a decision support tool should deliver a solution within this timeframe to accommodate the dynamic nature of air traffic.

A considerable amount of effort has been devoted to reduce the running time due to optimization (the term "running time" used throughout this paper refers to wall-clock time, and the computational efficiency discussed hereafter is measured by wall-clock time only). Bertsimas et al. used a computer with higher configurations to test the BSP benchmark, and the running time was reduced to 16 min as a consequence [7]. Rios and Ross from NASA Ames Research Center achieved further speedup by employing multithreaded programming [8]. But, the parallelism was limited to a standalone computer. More recently, a fine-grain Eulerian–Lagrangian model was developed and tested [9,10]. The running time decreased from 2 h to 6 min by employing parallel computing [11], which leverages commodity computers to build a relatively cheap yet powerful computing platform. The model runs faster by splitting the computations on a cluster of 10 Dell workstations. However, prototyping a multithreaded program requires extensive programming skills to deal with communication and synchronization between computers, which are often out of the scope of ATM researchers' domain knowledge. Cloud computing makes the use of distributed systems easier. The word "cloud" is a metaphor describing a networked memory and storage system [12]. A well-known framework of cloud computing is Apache Hadoop MapReduce, which is an open-source software developed for large-scale data processing on clusters of commodity computers [13]. Cloud computing leverages the computer power of networked hardwares. Computer clusters function as a high-end computer while managing distributed computing resources that are transparent to the developer. Programming a multithreaded application is as easy as writing a single-thread program. Cloud computing is typically used for datacentric applications, such as deoxyribonucleic acid sequence analysis [14], remote sensing image analysis [15], and power system analysis [16]. Huge datasets are digested by a large number of CPUs that run concurrently in parallel. As a result, massive parallelism is achievable.

The ATM community has taken the initiative to leverage the power of cloud computing. The Federal Aviation Administration (FAA) announced its cloud computing strategy in May 2012 in response to the Office of Management and Budget's "cloud first" policy [17]. In the announcement, the FAA set forth a roadmap to deploy its cloud infrastructure in 2014, which will continue to mature through 2016. This cloud will serve both internal and external users to enhance the NAS's performance. NASA also initiated a project in 2012. It worked with the General Electric Company to introduce the cloud environment into the Next Generation Air Transportation System [18]. The infrastructure changes are supposed to revolutionize the data storage model and software management. From a research perspective, the incoming changes mean an opportunity to reevaluate existing models that could potentially take advantage of the cloud platform to improve performance. This paper is in response to this call. A link transmission model (LTM), previously developed for addressing the NAS-wide traffic flow management problem, is revisited in this paper [9]. We refactor the code under the MapReduce framework to show that decomposable algorithms can be encapsulated in the cloud-based data processing model. We also show that analysis of the traffic data is well suited in the cloud environment.

The rest of this paper is organized as follows. Section II recapitulates the traffic flow management problem and the link transmission model. Section III briefly introduces the network construction from raw flight data. Section IV introduces the MapReduce programming model and how the LTM is coded under the framework. Section V presents a benchmark to show the improvement of efficiency. Section VI concludes this paper.

II. Link Transmission Model and Traffic Flow Management Problem

The link transmission model is a data-driven model. It establishes a route network based on radar tracks extracted from aircraft situation-distributed-to-industry (ASDI) data compiled by the ETMS [19,20]. Figure 1a is a snapshot of as-flown trajectories recorded in three months. These flights departed from Chicago O'Hare International Airport (ORD) and landed at Atlanta International Airport (ATL). Two clusters are clearly visible. This indicates that the aircraft follow predefined pathways with variations to meet traffic flow management (TFM) requirements. In Fig. 1a, flights in the left cluster stretched their flight paths to meet the estimated time of arrival imposed on the northwest gate of ATL. However, these flights largely traversed the same sector sequence. At the sector level, a route can be represented as a directed link sequence, with a link being an abstraction of passage within a sector, as shown in Fig. 1b. Trajectories that are not in any cluster due to significant deviations in response to a variety of stochastic events in the NAS do not contribute to the network abstraction.

In the LTM, air traffic is considered a discrete-time linear system. The variable $x_i^k(t)$ represents the number of flights in each link i on route k, and $q_i^k(t)$ is the outflow of this link. The flow must comply with the flow conservation principle:

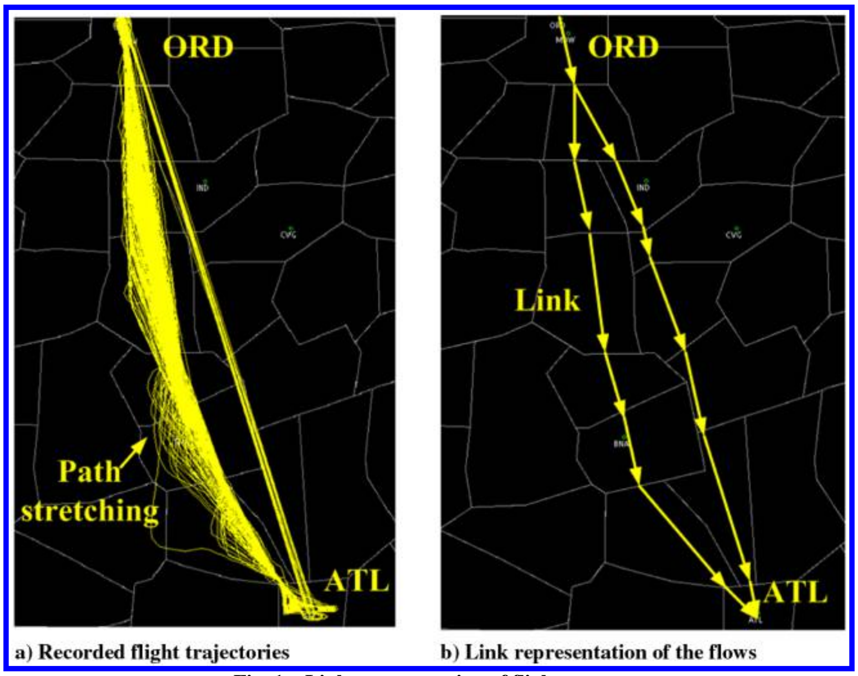
ORD
IND
CVG
Path stretching
ATL
a) Recorded flight trajectories
ORD
IND
CVG
Link
BNA
ATL
b) Link representation of the flows

Fig. 1 Link representation of flight routes.

$$x_i^k(t+1) = x_i^k(t) - q_i^k(t) + q_{i-1}^k(t), \quad \forall\, i \in \{0, \cdots, n^k\}, \qquad k \in \mathbb{K}, \qquad t \in \mathbb{T}$$

For the first link, its upstream outflow is departure $f^k(t)$. A basic traffic flow management problem is to control the flow rate $q_i^k(t)$ to ensure sector counts do not exceed sector capacity $C_s(t)$ while minimizing delays. This problem is formulated as an integer programming problem:

$$\min\ d = \sum_{t\in\mathbb{T}} \sum_{k\in\mathbb{K}} \sum_{1\le i\le n^k} c_i^k x_i^k(t) \tag{1}$$

subject to

$$x_i^k(t+1) = x_i^k(t) - q_i^k(t) + q_{i-1}^k(t) \tag{2}$$

$$\sum_{(i,k)\in Q_{s_i}} x_i^k(t) \le C_s(t), \qquad \sum_{(0,k)\in A_{\mathrm{arr}}} q_0^k(t) \le C_{\mathrm{arr}}(t), \qquad \sum_{(n^k,k)\in A_{\mathrm{dep}}} q_{n^k}^k(t) \le C_{\mathrm{dep}}(t) \tag{3}$$

$$\sum_{t\in\mathbb{T}} q_0^k(t) = \sum_{t\in\mathbb{T}} q_{n^k}^k(t) = \sum_{t\in\mathbb{T}} f^k(t) \tag{4}$$

$$\sum_{t=T_0^k+T_1^k\cdots+T_i^k}^{T_*^k} q_i^k(t) \le \sum_{t=T_0^k+T_1^k\cdots+T_{i-1}^k}^{T_*^k-T_i^k} q_{i-1}^k(t) \tag{5}$$

$$\sum_{t=0}^{T_0^k+T_1^k\cdots+T_i^k-1} q_i^k(t) = 0, \qquad x_i^k(0) = x_i^k \tag{6}$$

$$x_i^k(t) \in \mathbb{Z}_+, \qquad q_i^k(t) \in \mathbb{Z}_+ \quad \forall\, T_*^k \ge T_0^k + T_1^k\ \cdots\ + T_i^k, \qquad i \in \{0, \cdots, n^k\}, \qquad k \in \mathbb{K}, \qquad t \in \mathbb{T}, \qquad s \in \mathbb{S} \tag{7}$$

The objective function d minimizes the weighted total flight time of all flights in the planning horizon, which is equal to minimizing the total delays. Inequalities (2–7) regulate traffic flow behaviors. Inequality (3) enforces en route and airport capacity constraints. Equation (4) states that the total inflow into a route is equal to its total output. Inequality (5) dictates that every flight must dwell in a link for at least T_i^k min. Equation (6) and inequality (7) are initial states and integer constraints, respectively. We refer readers to [9] for detailed discussions of these constraints. In addition, the LTM represents a deterministic scheduling that is distanced from operation realities. However, the aggregate flow model is able to include stochastic inputs to account for weather impact or demand uncertainty [21]. The focus here is computational efficiency.

The outputs of the optimization are controlled flows along each route. Specifically, the vector $[x_1^k(t), x_2^k(t), \cdots, x_{n^k}^k(t)]$ represents the state of route k at time t. As t evolves, the states of the vector represent the movement of traffic flow. But, the LTM is an aggregate model that loses track of flight identifications, and the outputs of the optimization are not in an immediate form of executable commands that air traffic controllers can read. However, the results can be translated into flight-specific actions by a disaggregation process. Since the vector $[x_1^k(t), x_2^k(t), \cdots, x_{n^k}^k(t)]$ sets globally optimal states for route k, these states can be used as constraints for scheduling the flights on route k where variables are defined as ground delays and airborne delays associated with individual flights. The disaggregation process is discussed in detail in [19,22]. The outputs of this process are delays imposed on individual flights in each sector.

Although the formulation is simple, a LTM instance could be intractable when it covers a vast airspace. The LTM is essentially an IP problem typically solved by a branch-and-bound algorithm that is not in polynomial time. Even for the linear programming (LP) relaxations, the best known runtime complexity is $\mathcal{O}(N^{3.5}L)$, where N is the number of variables and L is the length of the data [23]. For NAS instances with at least tens of thousands of variables, the runtime complexity is thus at an order of trillion. The LP relaxations at this scale are not solvable, let alone the original IP problem. The size of the problem must be reduced in order to proceed. All constraints are defined by route, except for inequality (3). Routes passing through the same sector are coupled together by the sector capacity. By assigning Lagrange multipliers $\lambda_s(t)$, $\lambda_{(0,k)}(t)$, and $\lambda_{(n^k,k)}(t)$ to each of these constraints and adding them to the objective function, constraints pertaining to different routes can be separated such that each route is associated with an independent and smaller IP problem. As such, the large-scale problem is decomposed into a collection of small subproblems. The optimization proceeds by iteratively finishing two tasks. The first one solves all the subproblems. The second one updates Lagrange multipliers using a subgradient method based on the solutions of subproblems [24]. The process is summarized in Algorithm 1.

Algorithm 1 Decomposed TFM optimization

```
Step 1: $\forall\ k \in \mathbb{K}$, solve:
min $d^k(\lambda_s(t)) = \sum_{t\in\mathbb{T}}(\sum_{i=2}^{n^k-1}[c_i^k + \lambda_s(t)]x_i^k(t) + [c_1^k + \lambda_{(1,k)}(t)]x_1^k(t) + [c_{n^k}^k + \lambda_{(n^k,k)}(t)]x_{n^k}^k(t))$
    subject to Inequalities (2, 4–7)
Step 2: Update master problem and Lagrange multipliers:
    $d(\lambda_s(t)) = \sum_{k\in\mathbb{K}} d^k(\lambda_s(t)) - \sum_{t\in\mathbb{T}}\{\sum_{s\in\mathbb{S}} \lambda_s(t)C_s(t) + \sum_{(0,k)\in A_{\mathrm{arr}}} \lambda_{(0,k)}(t)C_{\mathrm{arr}}(t)$
    $+ \sum_{(n^k,k)\in A_{\mathrm{dep}}} \lambda_{(n^k,k)}(t)C_{\mathrm{dep}}(t)\}$
    $\lambda_s(t) \coloneqq \max\{0, \lambda_s(t) + \frac{1}{j+1}(\sum_{(i,k)\in Q_{s_i}} x_i^k(t) - C_s(t))\}, \forall\ t \in \mathbb{T}, s \in \mathbb{S}$
    $j = j + 1$
```

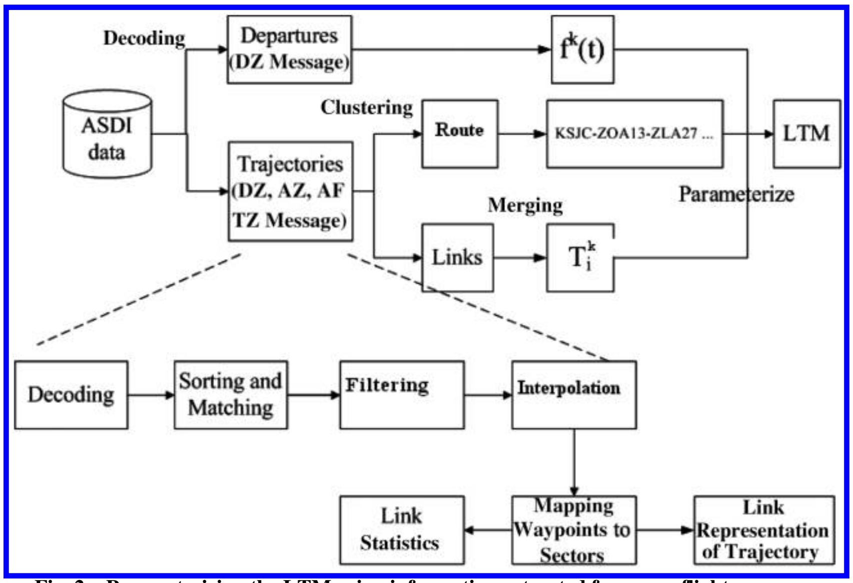

Fig. 2 Parameterizing the LTM using information extracted from raw flight messages.

III. Parameterization of the Link Transmission Model

To set up the LTM, the following information is needed: 1) routes identified in the NAS and the link sequence of each route, 2) the traversal time of each link T_i^k, and 3) scheduled departures $f^k(t)$. This information can be extracted from raw flight messages [25]. An ASDI log recording a full day of operations can be up to 500 MB. A snippet of the data looks like the following:

```
......
01F323201952KZHNTZ N717U/041 179 000 3627N/09305W
02F423201047KZAUFZ N398AC/251 LJ55/G 0440 MSN P2040 410 MSN..BAE..DJB..AGC/0115
02F523201049KZAUUZ USA123 T/B733/F 0432 4039N/09235W CLT./..LBF.SAYGE1.DEN/2203
......
```

Each line is a flight message sent by one of the facilities across the country. The most relevant messages include the departure message (DZ), arrival message (AZ), flight plan information (FZ), flow control track (TZ), and flight cancellation message. A trajectory can be restored by extracting message flow pertaining to the trajectory. But, the ETMS sorts incoming messages by timestamp. As a result, a trajectory's information could scatter through an ASDI file, especially for long-range flights. The whole file must be scanned to connect each piece of information pertaining to a flight. A flowchart of parameterization of LTM is shown in Fig. 2. To extract trajectories, the data have to be processed as follows:

1) The first process is decoding. ASDI compiles structured data [19]. Each message starts with a timestamp in conjunction with the message type, followed by aircraft identification (ACID), and a varying number of fields defined for specific message types. The flight message can be interpreted and stored in a list.

2) The second process is sorting and matching. Grouping the decoded flight messages by ACID yields a flight history of each aircraft.

3) The third process is filtering. ASDI data are flawed with mismeasurements [26]. Some waypoints deviate from a nominal flight path by big displacements, which would lead to erroneous flight time estimation in the subsequent steps. Therefore, each waypoint must be scanned and filtered to remove anomalies.

4) The fourth process is 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 must be scanned and interpolated if the interval between two successive waypoints is larger than 1 min. Interpolation is linear and based on the three-dimensional position and speed of the waypoint where interpolation begins.

5) The fifth process is sector mapping. Once filtered, the trajectory is represented by a vector of geographic coordinates. It must be translated into a link representation. Mapping coordinates to the sectors can be done by using the Ray casting algorithm [27], which is able to determine whether a point intersects a polygon.

6) The sixth process involves the link representation for the route. The individual traversal time T_i^k is obtained by calculating the difference of the entry record and exit record to/from a particular sector.

7) The seventh process involves the link statistics. The traversal time T_i^k is one of the most important parameters for the model setup. Given the defects inherent in ASDI data, estimating T_i^k based on individual trajectories is not reliable. For an estimate to be statistically significant, a large sample set is required. To identify a route, we stipulated that the sample size must be at least 70% of the number of days when the data were collected. In this paper, our dataset spans 84 days. A route was identified if there were at least 59 trajectory samples. To estimate the traversal time of a link, we took the mode of a sample set.

IV. MapReduce Programming Model

Apache Hadoop (high-availability distributed object-oriented platform) is the most popular open-source cloud environment[‡] used for large-scale processing of data. It is deployed on commodity hardware that is managed by the Hadoop distributed file system (HDFS), as shown in Fig. 3. In a Hadoop cluster, the master machine, also known as the name node in the context of the HDFS, automatically coordinates slave machines (i.e., data nodes) to manage the file system. Files are split into trunks, with multiple replicas stored across data nodes to guarantee accessibility in case of individual node failure. This design also minimizes data transfer by prioritizing the ingestion of local data on data nodes. The name node runs a

[‡]Data available online at http://hadoop.apache.org [retrieved 2015].

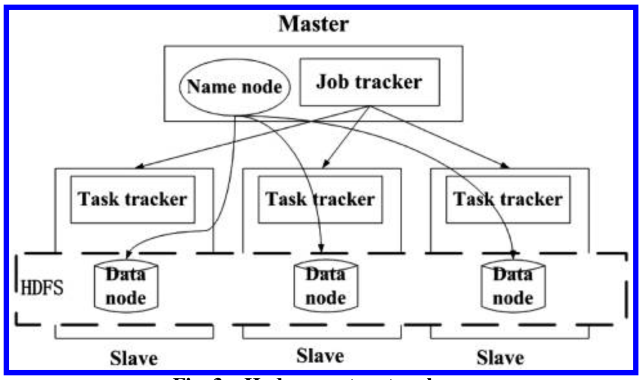

Fig. 3 Hadoop system topology.

daemon process called JobTracker that is responsible for job scheduling, and data nodes run daemon processes that are called TaskTracker. JobTracker coordinates TaskTracker to perform parallel computing.

Hadoop's MapReduce is a programming model written in Java for parallel list processing [28]. MapReduce centers around the concept of a $< key, value >$ pair, which is a formatted data structure. The key and value can be any data type. A MapReduce job is to sequentially digest a list of $< key, value >$ and generate another list of $< key'', value'' >$ as output. Figure 4 shows the data flow. Input files are first divided into multiple splits, with each being read into $< key, value >$ format; then, each split concurrently goes through two sequential phases:

1) The first phase is to map. Each of the key–value pairs $< k, v >$ is loaded into a *mapper* function, where they are mapped to a new intermediate set of key–value pairs $< k', v' >$ in accordance with user-specified rules. Once all key–value pairs are processed, the intermediate multisets are automatically sorted and shuffled with respect to k'. Pairs with the same key are aggregated, i.e., $< k', v'_1, v'_2, \cdots] >$.

2) The second phase is to reduce. Each aggregated pair is loaded into a *reducer* function, where $< k', v'_1, v'_2, \cdots] >$ is mapped to a new key–value pair $< k'', v'' >$ in accordance with a user-specified rule. These pairs are written into the output file as a list.

In general, the size of the output list is smaller than the size of the input list. Information is extracted from a large dataset through mapping operations. Users specify the mapping operations by implementing two Java classes: *mapper* and *reducer*. Their pseudocodes are given as follows:

```
mapper (IntWritable $k$, TextWritble $v$){ // k: line number. v: a line of input file
    $< k', v' >$ doSomething($k$, $v$)          // processing goes here, return an intermediate set
    Emit($k'$, $v'$) ;                          // $k'$ and $v'$ are user-defined data structures
}

reducer (Writable type $k'$, Writable set $[v'_1, v'_2, \cdots]$){
  for (each element $v'_i$ in $[v'_1, v'_2, \cdots]$ ) do{
    $< k'', v'' >$ doSomethingElse($k'$, $v'$)     // some other processing goes here:
    Emit($k''$, $v''$) ;
  }
}
```

MapReduce schedules multiple *mapper* or *reducer* instances running on Hadoop clusters in parallel to achieve high performance.

MapReduce's list-processing model makes it a perfect scheme for the LTM modeling. Particularly, MapReduce automatically sorts key–value pairs by key, which significantly reduces the programming workload when dealing with ASDI data decoding and trajectory analysis. Even solving TFM optimization can take advantage of the *mapper* and *reducer* functions. The LTM problem is developed in two stages. In the first stage, a large ASDI dataset is processed to prepare parameters for the LTM setup. The steps described in Sec. III.B are refactored in MapReduce jobs. In the second stage, the recursive optimization algorithm is encapsulated in a chain of MapReduce jobs. A high-level description of each job is provided next. For clarity, we denote the *mapper* function and *reducer* function as $\kappa()$ and $\rho()$, respectively; and we denote jobs in the first stage as "Job1.x" and jobs in the second stage as "Job2.x." In a job, the output of $\kappa()$ will be the input of $\rho()$. Jobs are concatenated in a job flow. The output of an upstream job is the input of the downstream job.

k: Offset from the first line of ASDI data file.

v: Raw messages, e.g., AF, DZ, TZ, etc.

k': ACID.

v': Flight information associated with message type, i.e., $(\varphi(t_m), \lambda(t_m), h(t_m))$ for TZ message or departure time for DZ message.

k'': ACID + t_{dep}.

v'': $[\varphi(t_m), \lambda(t_m), h(t_m)]$.

$< k', v' > \kappa(k, v)$: Read a line from the ASDI file. Decode the flight message. Emit flight information associated with ACID + t_{dep}.

$< k'', v'' > \rho(k', v')$: Correspond to the sorting and matching step. Since MapReduce automatically sorts the decoded messages by ACID, $\rho()$ receives a list of flight information associated with an ACID. To separate connecting flights, sort flight information by timestamp. Pair up DZ and AZ messages to set the time boundaries for a flight. Combine ACID with departure time t_{dep} as a unique identification for a flight. Emit matched waypoints to an output file.

k: Offset from the first line of output file derived from job 1.1.

v: Content of a line.

k': ACID + t_{dep}.

v': $[\varphi(t_m), \lambda(t_m), h(t_m)]$.

k'': ACID + t_{dep}.

v'': Interpolated $[\varphi(t_m), \lambda(t_m), h(t_m)]$.

$< k', v' > \kappa(k, v)$: Read a line from the output file from job 1.1. Emit waypoints associated with ACID.

$< k'', v'' > \rho(k', v')$: correspond to the filtering and interpolation steps. Derive a list of waypoints of a flight sorted by timestamp. Check anomaly and interpolate as necessary. Emit interpolated trajectory associated with ACID + t_{dep}.

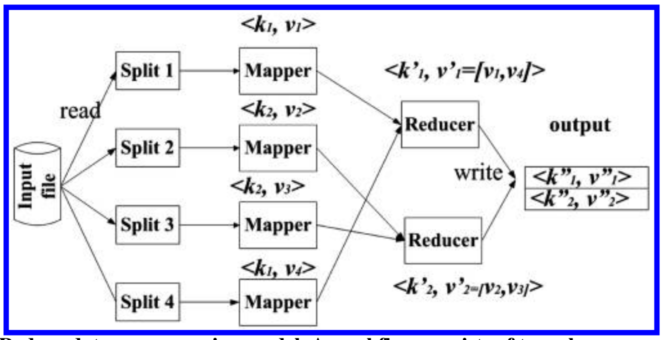

Fig. 4 MapReduce data programming model. A workflow consists of two phases: map and reduce.

k: Offset from the first line of output file derived from job 1.2.
v: Content of a line.
k': ACID + t_{dep}.
v': Sector (t_m).
k'': Link name.
v'': T_i^k.
$< k', v' > \kappa(k, v)$: Correspond to the sector mapping step. Read a line from the output file from job 1.2. Map waypoints to sectors. Emit sectors associated with ACID + t_{dep}.
$< k'', v'' > \rho(k', v')$: Correspond to the link representation of route and link statistics steps. Derive a list of sectors a flight traverses, and find out the links and associated traversal time. Emit link name and traversal time.
k: Offset from the first line of output file yielded in job 1.3.
v: Content of a line.
k': Link name.
v': T_i^k.
k'': Link name.
v'': $\hat{T}_i^k$.
$< k', v' > \kappa(k, v)$: Read a line from the output file derived from job 1.3. Emit link name and its traversal time T_i^k.
$< k'', v'' > \rho(k', v')$: Derive a list of traversal time associated with a link. Estimate the mode of the samples. Emit link name and its estimated traversal time $\hat{T}_i^k$.

Routes and links information is stored in a database for the setup of subsequent LTM optimization. In addition, the initial Lagrangian multipliers $\lambda_{s_i}(0)$ and scheduled departure $f^k(t)$ must be parameterized before kicking off a LTM optimization. The initial Lagrangian multipliers were set to be one. Since commercial carriers usually file their flight plans with the air traffic authority approximately 3 h before departure, $f^k(t)$ is obtained by analyzing FZ and AF messages, which provide proposed route and departure time. Routes extracted from ASDI data are represented by links. To match the route specified in a FZ message to a route stored in the database, all the waypoints and fixes specified in the FZ messages are mapped to sectors. These sectors in conjunction with the origin/destination airports are used to compare with routes in the database. The route with the most matched sectors is considered to be the proposed route. A flowchart is shown in Fig. 5, and the workflow is given as follows:
k: Offset from the first line of output file yielded in job 2.1.
v: Content of a line.
k': Sector s_i.
v': $x_i^k(t)$.
k'': Sector s_i.
v'': $\lambda_{s_i}(t)$.

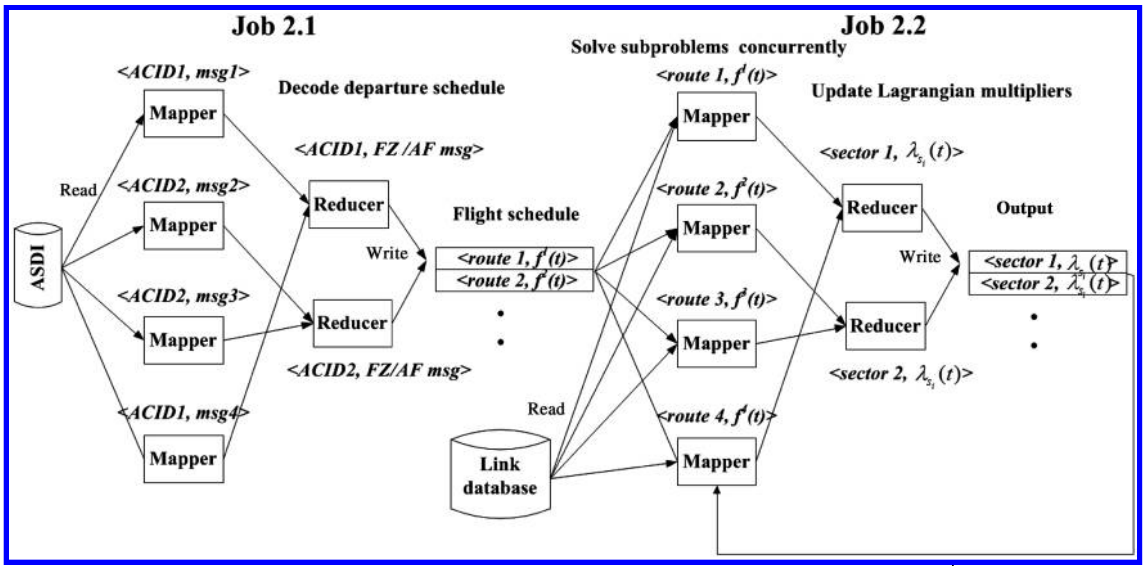

Fig. 5 Flowchart of the recursive algorithm based on MapReduce. Job 1.2 decodes DZ messages to parameterize $f^k(t)$. Job 2.2 recursively runs the optimizations.

Table 1 Extracted information from the ASDI data

Information	Each file on average	Total
ASDI file size	474 MB	39.8 GB
Flight messages	6,151,970	5.17×10^8
Flights	62,462	5,246,808
Sectors	1,121	1,258
Airports	3652 (in the U.S.)	3838 (in the U.S.)
	4013 (outside the U.S.)	4502 (outside the U.S.)
Routes	87,351	99,337
Links	26,175	29,586

$< k', v' > \kappa(k, v)$: Read a line from the output file derived from job 2.1. Retrieve link information from database and set up subproblem using time series $f^k(t)$, traversal time T_i^k, and initial Lagrangian multipliers $\lambda_{s_i}(t)$. Solve the subproblem. Emit sector name and the sector count $x_i^k(t)$ contributed by this route.

$< k'', v'' > \rho(k', v')$: Obtain a list of sector counts contributed by different routes. Aggregate the sector counts and update Lagrangian multipliers. Emit sector and its Lagrangian multipliers for next iteration.

V. Simulation Results

This section presents NAS-wide instances of TFM problems that run on a Hadoop cluster with six nodes. Each instance covers traffic in the continental U.S. airspace in a 2 h period. The Hadoop nodes were DELL workstations configured with an Intel i7 CPU and 16 GB of RAM. All workstations run Ubuntu 10.04 with Hadoop 0.20.2. Eighty-four ASDI data files corresponding to the traffic of 84 days were used to train parameters for the LTM. Table 1 summarizes the information culled from the data. Over 62,000 flights were tracked in the NAS each day, and over 6 million flight messages were filed. A large volume of data provides a statistically significant sample pool. As a prototype, we focused on the airspace in the continental United States only, which contains 1258 sectors. Airspace outside the continental Unites States was classified as an "international" sector where international flights originate from or are destined to. With this simplification, almost 30,000 links and 100,000 routes were identified. The route network includes 3838 general public airports in the United States. It covers the majority of commercial aviation routings in the U.S. airspace. Despite the large number of routes identified, only those involved in the planning horizon were pulled out of the database for the TFM problem setup.

To examine the efficiency of the cloud platform, we used both a standalone computer and a Hadoop cluster to process the ASDI data. In Hadoop, the ASDI files were loaded into the HDFS before parsing the flight messages; hence, the processing time does not account for the file transfer. We did likewise in the serial processing. Figure 6 shows the breakdowns of the running times of both serial and parallel computing. The improvement of efficiency is significant. The Hadoop cluster took 12.1 h to finish the processing with multiple threads, whereas the standalone computer took 94.5 h. The parallel computing was about eight times faster than the monolithic threading.

The TFM problem was validated by running a 2 h NAS-wide instance, which represents the high-traffic period of a day. Two-thousand, three-hundred, and twenty-six routes were involved in the scenario. The optimized traffic was implemented in NASA's simulation platform: Future ATM Concept Evaluation Tool (FACET) [29]. To test the model, we purposely reduced the sector capacity to 70% of the monitor alert parameters used in real-world operations [30]. The recorded traffic exceeded the reduced system capacity as a consequence. Figure 7 shows snapshots of the traffic simulated by FACET. We used different coloring to represent sector loads, indicating the number of aircraft exceeding the sector capacity, the number of aircraft close to the sector capacity, and the number of aircraft reaching half of the sector capacity. It is clear that the optimization alleviates congestion. The number of overloaded sectors were significantly reduced. There were still rare cases of overloading sectors. This is because we set a cap of 50 for the stopping criterion of iterations, which balances the quality of solutions and time that the users may invest. Indeed, the ATC may accept an imperfect scheduling delivered in a timely way instead of a perfect solution without time limit.

The running time of the optimization is shown in Fig. 8. As a configurable parameter, the maximum number of mappers allowed on a node can be specified by users as a means of controlling concurrency level. The running time decreases as more mappers are launched. But, the efficiency stops increasing when the number of mappers is more than eight. This is due to the fact that an Intel i7 CPU has four physical cores, with each being capable of handling two threads simultaneously. As a result, the maximum number of threads a node can run in parallel is eight. In theory, the Hadoop cluster can have at maximum $6 \times 8 = 48$ threads running concurrently; however, the speedup is not linear to the number of threads. Running the same TFM problem on a standalone computer took 135 min in comparison to 8.5 min on the Hadoop cluster. Only 16 times the speedup was achieved. There are two reasons to account for this. First, the parallel programming model has inherent overheads such as communication and synchronization between nodes. Second, unbalanced workloads cause idle time for some nodes. Figure 9 shows the running time of solving subproblems on each node. Node 3 is about 4.5 s ahead of node 4 in each iteration. In the implementation, we evenly distributed subproblems to each node. Although all the nodes have the same configuration, the complexity of subproblems has a wide range due to the length of routes. A more sophisticated dynamic allocation algorithm may help to reduce the gap; we leave this to our future work.

To evaluate the performance of the cloud-based TFM optimization, 360 NAS-wide instances were tested. We divided 30 days into 360 2 h periods. Each 2 h traffic was used to run the TFM optimization. These instances included both low- and high-traffic scenarios. Figure 10 shows the

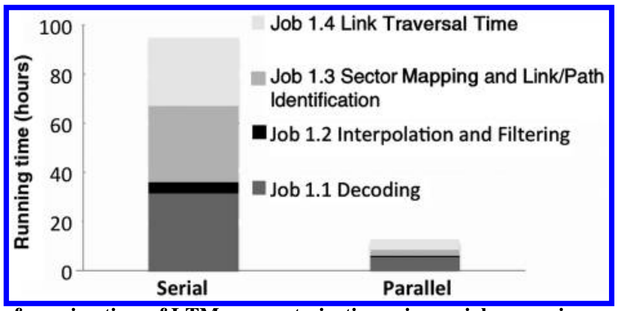

Fig. 6 Comparison of running time of LTM parameterization using serial processing and parallel processing.

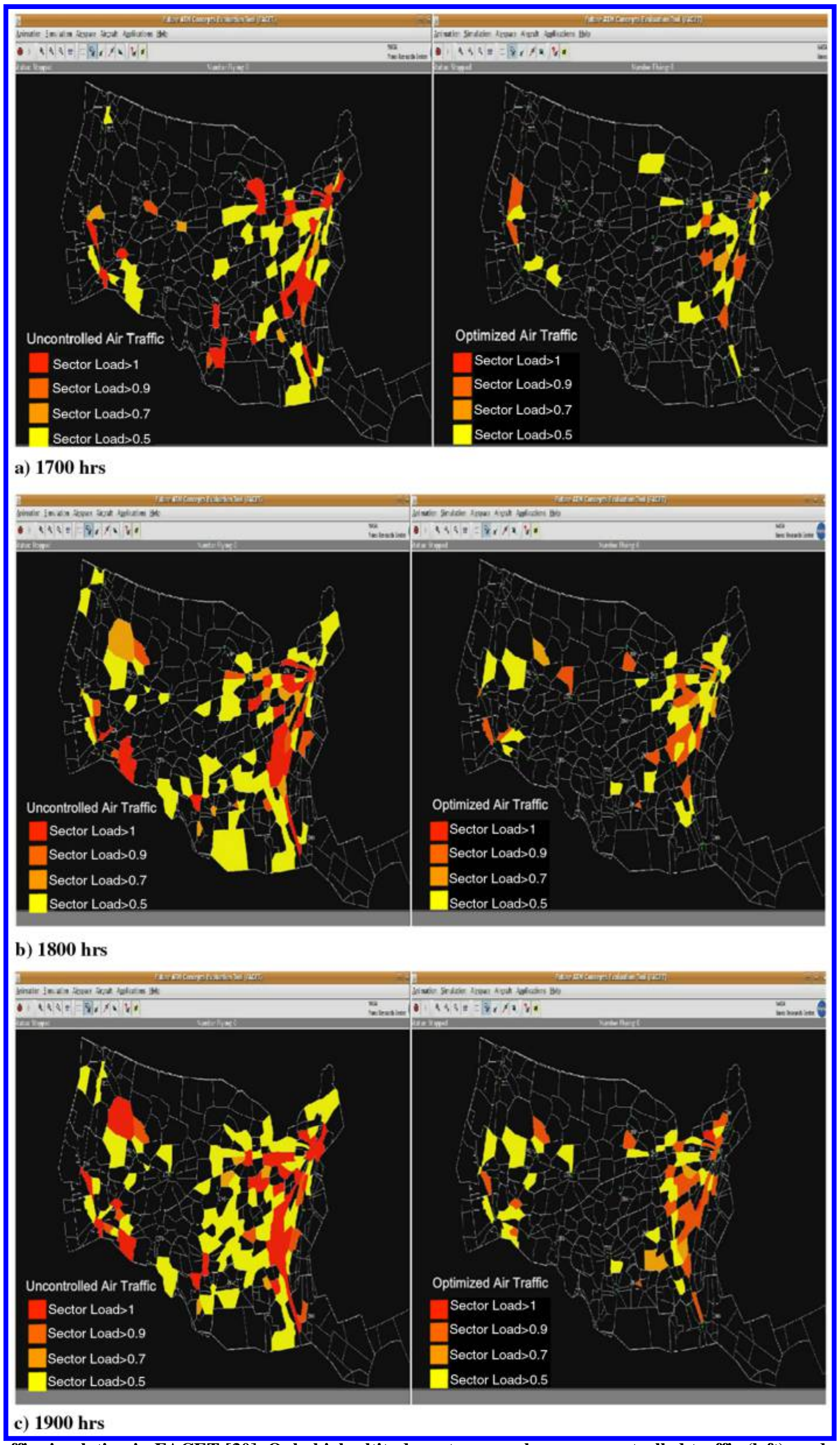

Fig. 7 Snapshots of traffic simulation in FACET [30]. Only high-altitude sectors are shown: uncontrolled traffic (left), and optimized traffic (right). Coloring represents the degree of sector load.

results. In general, the running time is proportional to the problem size. When more than 1500 routes are involved, the chance to solve a TFM problem in a few minutes is slim. However, the running time can be leveraged by deploying more nodes in the cluster.

An advantage of the MapReduce framework is its built-in fault-tolerance capability. Figure 11 shows a test where a node was purposely shut down during the iterations. It took longer to finish the MapReduce job in response to the failure of the node. As the master constantly communicates with the nodes, when it detects that a node fails to respond for a preset period of time, it reschedules the task to be reexecuted on other nodes so that the job can continue. As such, MapReduce recalculated the data splits and assigned jobs to the active nodes in the remaining iterations. As a result, the whole optimization was completed without interruptions. Fault tolerance is an important feature lacking in non-cloud-based computational frameworks.

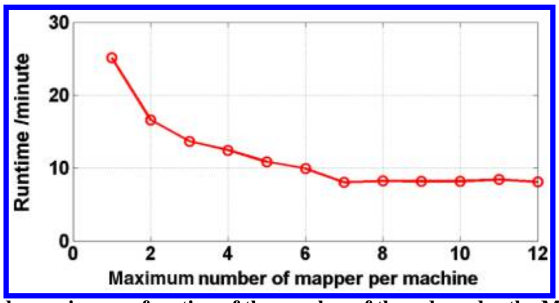

Fig. 8 Running time decreasing as a function of the number of threads under the MapReduce framework.

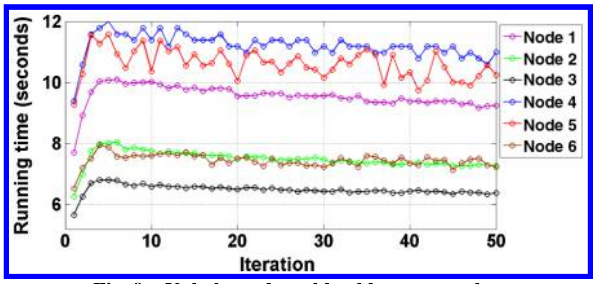

Fig. 9 Unbalanced workload between nodes.

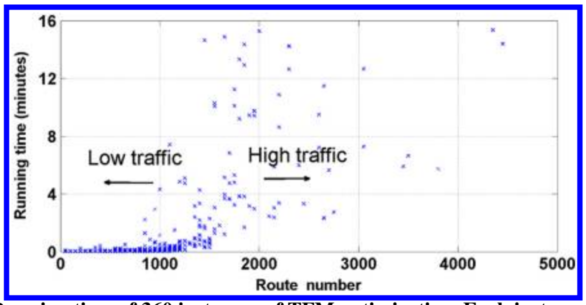

Fig. 10 Running time of 360 instances of TFM optimization. Each instance spans 2 h.

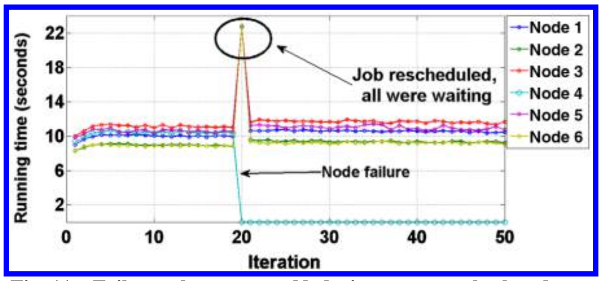

Fig. 11 Failure tolerance test. Node 4 was purposely shut down.

VI. Conclusions

This paper addresses the potential impact of a novel, cloud computing platform solving large-scale optimization problems that are traditionally intractable. In the application for air traffic management, this platform is potentially helpful for developing supporting tools for operational decision making at a national level, such as in the FAA System Command Center. It is demonstrated that traffic flow management problems can be solved more efficiently by breaking down large problems into smaller problems using Lagrangian methods and employing parallel programming model. MapReduce's programming model saves developers from dealing with multithreaded programming work, significantly reducing development workload. Users can customize their applications with the MapReduce library that comes with the Hadoop package. This framework is not only efficient but also robust. Its distributed file system enhances data safety as well as runtime fault tolerance. All these features make it a suited platform for air traffic management research. In the long run, shifting the ATM concept evaluation to the cloud environment will speed up the development cycle and reduce the cost. More important, the cloud provides a hardware solution that is scalable to the problem size, helping researchers to tackle complicated computationally intensive aviation problems.

Acknowledgments

This material is based upon work supported by the National Science Foundation under grant no. 1035532. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

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

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

[3] Tien, S., Taylor, C., Zhou, Y., Wan, Y., and Wanke, C., "A Route-Based Queuing Network Model for Air Traffic Flow Contingency Management," *11th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, AIAA Paper 2011-6863, Sept. 2011.

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

[5] Sridhar, B., Soni, T., Sheth, K., and Chatterji, G. B., "Aggregate Flow Model for Air-Traffic Management, Journal," *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 4, July–Aug. 2006, pp. 992–997.
doi:10.2514/1.10989

[6] Bertsimas, D., and Stock-Petterson, S., "The Air Traffic Flow Management Problem with En route Capacities," *Operations Research*, Vol. 46, No. 3, 1998, pp. 406–422.
doi:10.1287/opre.46.3.406

[7] Bertsimas, D., Lulli, G., and Odoni, A. R., "The Air Traffic Flow Management Problem: An Integer Optimization Approach," *Proceedings of the 13th International Conference on Integer Programming and Combinatorial Optimization*, Vol. 46, Springer, New York, 2008, pp. 35–46.

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

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

[10] 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, May 2008, pp. 616–628.
doi:10.2514/1.31717

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

[12] White, T., *Hadoop: The Definitive Guide*, Yahoo! Press, Sebastapool, CA, June 2009, p. 41, Chap. 3.

[13] Dumbill, E., *Planning for Big Data: A CIO's Handbook to the Changing Data Landscape*, O'Reilly, Sebastapool, CA, March 2012, Chap. 3.

[14] Hosny, A. M., Shedeed, H. A., Hussein, A. S., and Tolba, M. F., "Cloud Statistical Significance Estimation for Optimal Local Alignment of Huge DNA Sequences," *8th International Conference on Informatics and Systems (INFOS)*, IEEE Publ., Piscataway, NJ, May 2012, pp. 48–55.
doi:10.1002/cpe.2953

[15] Almeer, M. H., "Cloud Hadoop Map Reduce for Remote Sensing Image Analysis," *Journal of Emerging Trends in Computing and Information Science*, Vol. 3, No. 4, April 2012, pp. 637–644.

[16] Huang, Q., Zhou, M., Zhang, Y., and Wu, Z., "Exploiting Cloud Computing for Power System Analysis," *2010 International Conference on Power System Technology (POWERCON)*, IEEE Publ., Piscataway, NJ, Oct. 2010, pp. 1–6.
doi:10.1109/POWERCON.2010.5666051

[17] "FAA Cloud Computing Strategy, Final: Ver. 1.0," Federal Aviation Administration, Washington, D.C., 2012.

[18] Kulkarni, D., "General Electric Begins Work on Cloud Computing for Air Traffic Management," *NASA Intelligent Systems Division News*, http://ti.arc.nasa.gov/news/ge-cloud-atc/ [retrieved June 2014].

[19] "Aircraft Situation Display to Industry: Functional Description and Interface Control Document," Ver. 4.0, Volpe Center Automation Applications Division, Rept. ASDI-FD-001, Cambridge, MA, 2000.

[20] "Enhanced Traffic Management System (ETMS)," U.S. Dept. of Transportation, National Transportation Center Volpe, TR VNTSC-DTS56-TMS-002, Cambridge, MA, 2005.

[21] Bosson, C. S., and Sun, D., "An Aggregate Air Traffic Forecasting Model Subject to Stochastic Inputs," *AIAA Guidance, Navigation, and Control and Co-Located Conferences*, AIAA Paper 2013-5032, Aug. 2013.

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

[23] Ye, Y., "An $\mathcal{O}(n^3L)$ Potential Reduction Algorithm for Linear Programming," *Mathematical Programming*, Vol. 50, No. 2, 1991, pp. 239–258.
doi:10.1007/BF01594937

[24] Sun, D., Clinet, A., and Bayen, A., "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

[25] Alexander, K., "Computation and Analysis of Aircraft Proximities in the NAS Using ETMS Data," *24th Digital Avionics Systems Conference*, Vol. 1, IEEE Publ., Piscataway, NJ, Oct. 2005, pp. 3.B.6-1–3.B.6-11.
doi:10.1109/DASC.2005.1563346

[26] Salaun, E., Gariel, M., Vela, A. E., and Feron, E., "Airspace Complexity Estimations Based on Data-driven Flow Modeling," *AIAA Guidance, Navigation, and Control Conference*, AIAA Paper 2010-8074, Aug. 2010.
doi:10.2514/6.2010-8074

[27] 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 University of Technology, Smolenice, Slovakia, April–May 2012.

[28] Lammel, R., "Google's MapReduce Programming Model: Revisited," *Science of Computer Programming*, Vol. 70, No. 1, Jan. 2008, pp. 1–30.

[29] Bilimoria, 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.

[30] "Facility Operation and Administration," U.S. Dept. of Transportation, Federal Aviation Administration, Order JO 7210.3W, Feb. 2010.

A. Banerjee
Associate Editor