A Parallel Computing Framework for Large-scale Air Traffic Flow Optimization

Yi Cao and Dengfeng Sun

***Abstract*—Optimizing the nationwide air traffic flow entails computational difficulty as the traffic is generally modeled as a multi-commodity network which involves a huge number of variables. This paper presents a framework which speeds up the optimization. The nationwide air traffic is modeled using a Link Transmission Model, to which a dual decomposition method is applied. The large-scale problem is decomposed into a master problem and a number of independent subproblems which are easy to solve. As a result, the execution of solving the subproblem is parallelizable. A parallel computing framework is based on multiprocessing technology. The master problem is updated on a server, and a client cluster is deployed to finish the subproblems such that the most computationally intensive part of the optimization can be executed in parallel. The server and the clients communicate via TCP/UDP. An adaptive job allocation method is developed to balance the workload among each client, resulting in maximized utilization of the computing resources. Experiment results show that, compared to an earlier single process solution, the proposed framework considerably increases the computational efficiency. The optimization of a 2-hour nationwide traffic problem involving 2326 subproblems takes six minutes using 10 Dell workstations. The increased computation workload due to increased number of subproblems can be mitigated by extension of computer delpoyment.**

***Index Terms*—National Aerospace System (NAS), air traffic flow optimization, large-scale optimization, parallel computing.**

I. Introduction

THE National Airspace System (NAS) is monitored and controlled by the Air Traffic Control System Command Center where traffic managers generally look at the holistic traffic four hour into the future. There are various automation tools to assist the air traffic controllers in decision-making process, but few are designed for coordinating the traffic flows at a national level. Tractability and exponential increase of runtime represent the difficulty in developing efficient model for this purpose.

In air Traffic Flow Management (TFM) literature, the Bertsimas and Stock-Patterson (BSP) model is among the most recognized paradigms developed to address the TFM problem [1]. The scheduling of a flight is determined by a binary vector that is subject to capacity constraints. Compared to models where the dynamics of traffic flows are governed by partial differential equations [2], the BSP is easy to be solved by Linear Programming (LP). But the BSP is a Lagrangian model whose dimension linearly increases against the aircraft number. Menon et al. developed an aggregated model which formulates the NAS traffic in a framework of longitude-latitude tessellation using the Eulerian approach [3]. This model was initially designed for traffic prediction, and later extended to be a close-loop controller [4]. The dimension of the Menon Model is correlated to the geographic coordinate resolution. A typical setting is $1^\circ \times 1^\circ$ for a surface element. For the NAS traffic where the target airspace covers a vast area, computational issue arises from the large-scale TFM problem.

Efforts to address the NAS-wide TFM problem can be found in [5] and [6]. Taking advantage of the block-angular form of the BSP, Rios et al. used Dantzig-Wolfe decomposition to parallelize the model. Flights are grouped into smaller subproblems so that each subproblem can be solved independently. Sun et al. developed a Large-capacity Cell Transmission Model (CTM(L)) [9] and parallelized the model using the dual decomposition method. The traffic is formulated as a multi-commodity network and flights are grouped into flows in terms of origin and destination pairs. Based on the dual decomposition method, the traffic is decomposed path by path, and each flight path is optimized independently. As a result, the dimension of CTM(L) depends on the number of flight paths identified in the NAS. Both Dantzig-Wolfe decomposition and dual decomposition method make the large-scale TFM problem solvable. However, the problem associated with previous works is that the TFM problem is formulated using Mixed Integer Linear Programming (MILP) but solved using Linear Programming (LP) relaxation. A rounding heuristic must follow to obtain an integer solution. As such, optimality are guaranteed. Such compromise is made to avoid long runtime of solving MILP. In [6], it is shown that a 2-hour NAS-wide TFM optimization can be done within 149 seconds. In [10], a Link Transmission Model (LTM) is derived by coarsening the flight path elements used by CTM(L). The best complexity for solving a LP is $O(N^{3.5}L)$ [11], where N is the number of variables and L is the data length. The LTM reduces the number of variables by an order of ten, thus dramatically decreasing the computational complexity. When dealing with the NAS-wide TFM problem, LTM still resorts to the dual decomposition method. The MILP is directly solved by a CPLEX Integer Program solver. Compared to solving a LP relaxation, it is more computationally expensive to obtain an integer solution. A 2-hour NAS-wide TFM problem takes up to two hours. The time consuming process prevents the model from serving as an efficient decision support tool.

So far, few prototypes capable of optimizing the nationwide air traffic flow with high efficiency have been reported. Rios et al. developed a parallelized BSP based on a client-server

Yi Cao is a PhD student with School of Aeronautics and Astronautics, Purdue University, West Lafayette, IN 47906-2045 USA. e-mail: cao20@purdue.edu.

Dengfeng Sun is a faculty member with School of Aeronautics and Astronautics, Purdue University, West Lafayette, IN 47906-2045 USA.

Manuscript received September 6, 2011; revised March 1, 2012.

architecture [12], where a server coordinates a number of solvers which handle traffic in each Center. The solvers and the server are instantiated with multiple threads. The threads communicate via socket and message queue. But the parallelism is performed on a single machine. Thus the advantage of the parallel algorithm is still leashed. Work presented in this paper is motivated by a desire to reduce the runtime and to expedite evaluation of NAS-wide TFM problems. A parallel computing framework is proposed which takes full advantage of distributed computing resources to finish the computationally intensive work. General purpose computers are used to tackle large-scale problems, which is significant to the nationwide traffic flow research. We develop a balancing algorithm to maximize the utilization of CPU time. The algorithm is characterized by straightforward calculation which causes little overhead, thus is cost-efficient. Moreover, the framework is able to cope with TFM problem at any scale as long as there are sufficient hardware. The LTM-based TFM problem has been closely examined in our previous papers [10], [13]. As a continuing effort, this paper focuses on how to parallelize the model and deploy it to distributed computing resources to achieve high computational efficiency.

The rest of this paper is organized as follows. Section II briefly introduces the Link Transmission Model and its decomposed form. The parallel computing framework is developed in Section III, including hardware deployment and software design. The bottleneck impeding the speedup of optimization will be identified and a workload balancing algorithm for maximizing the utilization of the cluster will be developed in this section. Section IV presents the experiment results. Computational efficiency is analyzed. Concluding remarks are given in Section V.

II. Formulation of the NAS traffic

A. Link Transmission Model

The Link Transmission Model uses the high altitude sectors as the geographic basis. A sector is a basic airspace session that is monitored by one or more air traffic controllers. Each flight path (referred to as path hereafter) is a flight course connecting a departure airport and an arrival airport, as shown in Fig. 1. The path is segmented by sector boundaries, each segment is a *link*. The travel time of a link is extracted from historical flight data. Aircraft are assumed to fly in the same speed when they pass through a specific link. There are tens of thousands of aircraft traveling on their paths throughout the day, forming a multi-commodity network. The NAS is covered by a huge network consisting of a large number of paths.

The state variable $x_i^k(t)$ is defined as the aircraft count in link i on path k at time t, which is an integer. The dynamics of the traffic flow is governed by the flow conservation, and is described as a discretized linear time varying system:

$$\begin{aligned} x_i^k(t+1) &= \beta_{i-1}^k(t)x_{i-1}^k(t) + (1-\beta_i^k(t))x_i^k(t) \\ x_0^k(t+1) &= f^k(t) + (1-\beta_0^k(t))x_0^k(t) \end{aligned}$$

where $f^k(t)$ is the scheduled departure into path k. $\beta_i^k(t)$ is a transmission coefficient representing the outflow from upstream link i to downstream link $i+1$, hence $0 \le \beta_i^k(t) \le 1$.

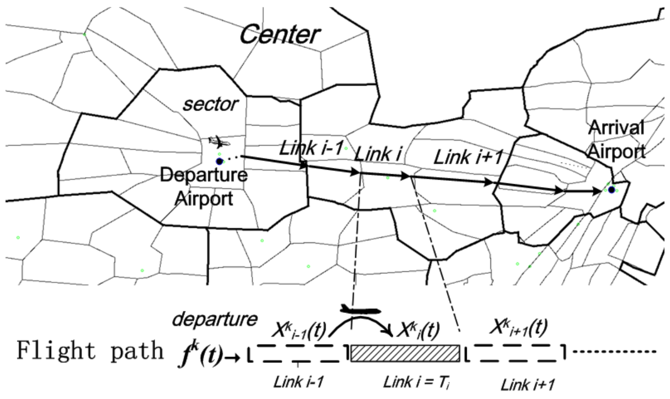


Fig. 1. Link Transmission Model.

In the TFM problem, $\beta_i^k(t)$ controls the flow rate such that the congestion of downstream link can be mitigated. In order to obtain a linear program, we use substitution $q_i^k(t) = \beta_i^k(t)x_i^k(t)$. By definition, $q_i^k(t) \le x_i^k(t)$, then the dynamics for path k can be described by the following state space:

$$\boldsymbol{X^k(t+1)} = \boldsymbol{IX^k(t)} + \boldsymbol{BQ^k(t)} + \boldsymbol{C}f^k(t) \tag{1}$$

where $\boldsymbol{I}$ is a $n^k \times n^k$ identity matrix, and n^k is the number of links on path k. $\boldsymbol{X^k}(t) = [x_0^k(t), x_1^k(t), \cdots, x_{n^k}^k(t)]^T$, $\boldsymbol{Q^k}(t) = [q_0^k(t), q_1^k(t), \cdots, q_{n^k}^k(t)]^T$, and

$$\boldsymbol{B} = \begin{bmatrix} -1 & 0 & \cdots & 0 \\ 1 & -1 & & 0 \\ & & \ddots & \\ 0 & \cdots & 1 & -1 \end{bmatrix}, \quad \boldsymbol{C} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

$$Q^k(t) \le X^k(t), \tag{2}$$

$$q_i^k(t), \quad x_i^k(t) \in \mathbb{Z}^+. \tag{3}$$

Constraint (2) is implicitly included in the system dynamics (1). But such redundant formulation is found to be beneficial to the convergence of the solution using the dual decomposition method. With the dynamics of the traffic at hand, we are ready to formulate the TFM problem.

B. Formulation of the TFM problem

A typical objective associated with the TFM problem is to minimize the total flight time over all aircraft in the planning time horizon T, which reflects the realistic goal to minimize fuel consumption:

$$\min \sum_{k=0}^{K}\sum_{t=0}^{T}\sum_{i=0}^{n^k} c^k x_i^k(t) \tag{4}$$

where K is the number of paths identified in the NAS. c^k is the weight imposed on path k.

The NAS currently uses *Monitor Alter Parameter* (MAP) as the sector capacity [14], alert will be issued if the sector is overloaded. Hence the sector count should not exceed the

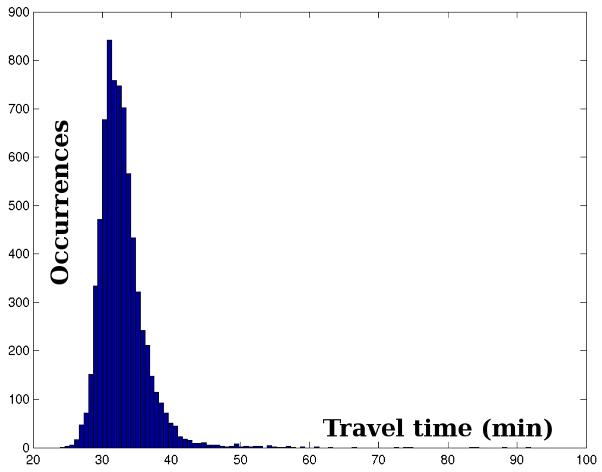


Fig. 2. A typical distribution of travel time for a link. Data extracted for a full year operations.

MAP value C_{s_i}:

$$0 \leq \sum_{(i,k)\in Q_{s_i}} x_i^k(t) \leq C_{s_i} \tag{5}$$

where Q_{s_i} represents the set of links lying in sector s_i.

Although aircraft travel in different speed, each aircraft must stay in a link for a certain amount of time. In order to guarantee a reasonable flow rate, a minimum travel time is imposed on each aircraft. We assume that the minimum travel time T_i is the average travel time that an aircraft needs to pass through link i, which is statistically derived from historical flight data [9]. A typical distribution of travel time for a link is shown in Fig. 2. The distribution is concentrated, thus an average number is reasonable to characterize the flow property in this link. The minimum travel time constraint is translated into the following inequality:

$$\sum_{t=T_0+T_1\ldots+T_i}^{T^*} q_i^k(t) \leq \sum_{t=T_0+T_1\ldots+T_{i-1}}^{T^*-T_i} q_{i-1}^k(t) \tag{6}$$
$$T^* \in \{T_0+T_1\cdots+T_i, T_0+T_1\cdots+T_i+1,\ldots,T\}$$

Equation (6) indicates that the accumulated inflow of a link at any instant t is greater than its accumulated outflow at instant $t+T_i$. This amounts to detaining an aircraft in link i for at least T_i.

In order to be in compliance with the flow conservation, namely the accumulated departures must equal the accumulated arrivals, we enforce the following constraint:

$$\sum_{t=0}^{T} q_{n^k}^k(t) = \sum_{t=0}^{T} f^k(t) \tag{7}$$

However, the NAS carries continuous traffic flows all the time. There are aircraft still being airborne at the end of the time horizon. As a result, Constraint (7) does not hold true for these aircraft. An adjustment can be made to accommodate these aircraft. Given a flght plan and the average travel time of each link, one is able to predict the nominal arrival time of an aircraft at any link along its path. If an aircraft is predicted to be airborne at the end of the time horizon, this aircraft is separated from its nominal path. Its destination is changed to be link $\hat{n}^k$ where it is predicted to arrive at T. This "shortened" path is treated as a special path k' which is subject to a variation of constraint (7):

$$\sum_{t=0}^{T} q_{\hat{n}^k}^{k'}(t) = 1 \tag{8}$$

Equations (1)-(8) formulate the TFM problem. Solution to this problem is the optimal traffic flow as well as the flow control for each path. Given that real traffic control is generally applied to individual aircraft rather than a flow, the flow control obtained from this model seems impracticable. In [15], a disaggregation method is developed to convert the flow control into flight-specific control actions for the CTM(L), which can be easily adapted to the LTM as well. But this is out of the scope of this paper.

C. Dual Decomposition Method

Equation (4) indicates that the number of state variable is determined by three indices: K, T, n^k. A long path generally passes through tens of sectors. There are approximately thousands of state variables for the path. Solving problem at this scale is not difficult for most software packages. However, if the traffic in the entire NAS is considered, the total number of state variables could be up to millions. For example, a 2-hour TFM problem involves approximately 2400 paths, and each path consists of 15 links on average, then there are $120 \times 2400 \times 15 = 4,320,000$ state variables. Moreover, $Q^k(t)$ is also treated as state variable at the same order as $X^k(t)$, then the number of state variables is up to 8,620,000. As such, even the most up-to-date optimizer is unable to handle a problem at this order. However, all constraints are separable in terms of path except for Equation (5). State variables of different paths are coupled only by the sector capacity constraint. In large-scale optimization problems, Lagrangian multiplier is often used to separate variables so that the problems can be decomposed [8]. More recently, Eun et al. used Lagrangian dual decomposition method to tackle the arrival scheduling problem which is known to be NP hard [7]. In [6], by means of Lagrangian multipliers, the sector capacity constraints are incorporated into the objective function. Then the formulation can be rearranged in terms of path. Each path is a smaller subproblem. For brevity, we refer readers to [13] for detailed derivation of the decomposed form of the TFM problem. The master problem and subproblem are directly given here:

Subproblem for path k:

$$d^k(\lambda_{s_i}^j(t)) = \min \sum_{t=0}^{T}\sum_{i=0}^{n^k}[c^k + \lambda_{s_i}^j(t)]x_i^k(t) \tag{9}$$

s.t. Equations (1) – (7)

master problem:

$$d(\lambda_{s_i}^j(t)) = \max\{-\sum_{t=0}^{T}\sum_{s_i=0}^{S}\lambda_{s_i}^j(t)C_{s_i} + \sum_{k=0}^{K} d^k(\lambda_{s_i}^j(t))\} \tag{10}$$

$$g_{s_i}^{j+1}(t) = -(\sum_{(i,k)\in Q_{s_i}} x_i^k(t) - C_{s_i}) \tag{11}$$

$$\lambda_{s_i}^{j+1}(t) := (\lambda_{s_i}^j(t) - \alpha^{j+1}g_{s_i}^{j+1}(t))_+ \tag{12}$$

where $\lambda_{s_i}^j(t)$ is the Lagrangian multiplier for j^{th} iteration for sector s_i. $g_{s_i}^j(t)$ is the subgradient direction. $\alpha^j = \frac{1}{j+1}$ is the subgradient step. j is the iteration index. $(\cdot)_+$ denotes retaining the non-negative value of a number.

A flowchart of the algorithm is shown in Fig. 3. The whole problem is decomposed path by path via the dual decomposition. Each subproblem is a small and independent optimization that is relatively easy to solve. The optimization progresses in an iterative manner to search the global optimum. The iteration is terminated if the change of the master problem is less than a preset value. Obviously, the computational efficiency can be improved if the subproblems are solved in parallel.

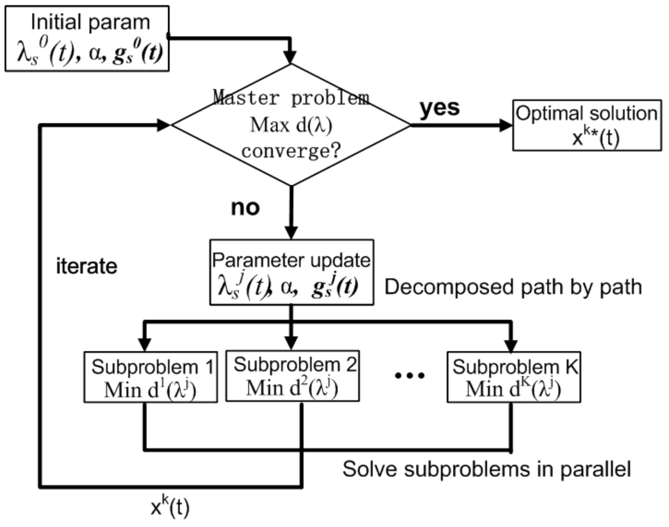


Fig. 3. Flow chart of the dual decomposition algorithm.

III. Parallel computing framework development

The TFM problem can be parallelized at two levels. First, the subproblems can be distributed to multiple computers which are connected in a network; Second, for each computer, the subproblems can be furthered distributed to multiple processes with each running an optimizer instantiation. CPLEX, one of the most prevalent optimization tool, supports multi-processing as well as Integer Programming. The subproblems can be solved by as many optimizer instances as the license permits. However, if the number of processes exceeds the number of processors, increasing the number of processes is not more than overloading the CPU. This section describes a platform for the parallel computing.

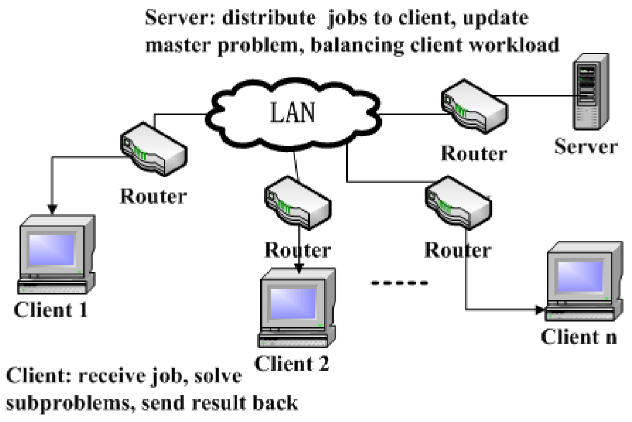

Server: distribute jobs to client, update master problem, balancing client workload
LAN
Router
Server
Router
Router
Router
Client 1
Client 2
Client n
Client: receive job, solve subproblems, send result back

Fig. 4. System topology.

A. Hardware deployment

In cybernetic terms, the master problem-subproblems structure fits the client-server model, as shown in Fig. 4. The server is responsible for allocating subproblems to the clients, updating the master problem and controlling the iteration process. Each client simply concentrates on solving subproblems. The client-server model can be connected via local area network (LAN), which provide the following advantages:

- Programmer can focus on application design and relegate the data transmission to the TCP/IP or UDP/IP protocol;
- Easy to deploy with existing network;
- High data throughput.

Paradigms like Parallel transportation Management System and High-Speed Railway Systems [16], [17], [18], have demonstrated that Internet-based structure can provide transportation systems with reliable connection.

B. Software design

A client-server software architecture is depicted in Fig. 5. To accelerate the computation, both *Server* and *Client* run multiple processes. Each process is responsible for different missions, e.g. communication, synchronization, solving subproblems. Details are provided as follows.

On the *Server* side, there are three processes.

- A *Master process* controls the progress of the whole optimization. When each iteration is completed, the *Master process* updates the Lagrangian Multipliers $\lambda_{s_i}^j(t)$ using the feedback data and determines the termination of the optimization. If the iteration should continue, the *Master process* allocates the subproblems to the clients which are commensurate to their computing power.
- A *Communication process* is responsible for sending and receiving data to/from other computers. It monitors the data transfer request from both internal *Master process* and external clients. When a sending request is received from the *Master process*, the *Communication process* will send the data buffered in the memory. On error, it will try again later until the data are successfully sent. When the clients have data to send back, the *Communication process* will first check whether there are unread data in

the buffer. If the buffer is ready, it conducts data buffering and notifies the *Master process* of new incoming data; if not, it simply rejects the data transfer request. The client will try again later until the data are successfully sent. Hence, it is the *Communication process's* responsibility to guarantee reliable data sending/receiving.

- A *Database process* provides an access to the database where the flight data are stored. To set up the LTM, information of links, paths and sectors is needed. All the information is calculated offline based on historical flight records. The *Master process* sends inquiries to the *Database process*, and the *Database process* responds by buffering the results to a shared memory.

The client has a similar configuration as the server.

- The *Communication process* is as same as the server's.
- A *Allocator process* is a counterpart of the *Master process*. It controls the optimization process on the client. It receives a batch of subproblems from the server and redirects the jobs to the *Child processes* for processing. Once a *Child process* finishes a subproblem and becomes idle, the *Allocator process* will send a new subproblem to it. The *Child processes* are kept busy, therefore the utilization of CPU is maximized.
- Several *Solver processes* concentrate on performing the optimization. Each *Solver processes* instantiates a CPLEX solver to optimize the subproblems. Given that each process would compete for the CPU time, the number of *Solver processes* should match up to the client's multiprocessing capability. Large number of processes inevitably incurs excessive overheads.

As the optimization is the most computation-intensive part, it is desirable to allocate the CPU time to the *Solver processes* as much as possible. Therefore, other processes periodically check incoming request and respond. If there is no request, they just sleep.

In a multiple processing and multiple computer environment, communication is crucial to achieving an efficient operation. Therefore, considerations are given to the following aspects.

- Internal communication. Processes need to communicate short messages as well as large trunk of data throughout the optimization process. For example, the *Master process* need to inform the *Communication process* of new data to send; The *Solver processes* need to inform the *Allocator process* that the subproblem is solved. Since the work of this framework is simple, a 32 bit integer is sufficient to encode the predefined actions that the server or clients will take. In this work, message and data are exchanged via shared memory, which enables different processes to directly read/write a block of memory. Each message or data trunk includes a sender ID and a receiver ID such that a process can identify the right message delivered to it when visiting the shared memory.
- External communication. The server and the clients communicate in a request-response mode. Whenever one sends a service request, the other responds with a predefined action. Computers communicate via *Socket*. Two data links are established. One is for exchanging short messages, and the other is for large data transfer. Short message is used for synchronization. For example, the server sends a message to the clients to start or stop an iteration. Since short messages must be received correctly, the message link uses Transmission Control Protocol (TCP) which provides reliable connection. Given that the server and clients need to frequently exchange parameters throughout the optimization process, such as the path information from the database, Lagrangian multipliers, and the optimized traffic flow, large data should be transferred quickly. The data link uses User Datagram Protocol (UDP) which is connectionless. The reliability of data transfer is guaranteed by the *Communication process*. A *Handshake* procedure is established to ensure that both sender and receiver are ready for data transfer. The sender first sends a send data request to the receiver via short message. Then the receiver will check whether the data in the data buffer have been read or not. If read, the data buffer is ready for new data, and the receiver responds with an approval message; If not been read, the receiver will reject the send data request with a reject message. Upon approval message, the sender begins sending data. The receiver sends back a confirmation message when the data transfer is finished. A checksum is appended at the end of each data package such that error checking can be performed by the receiver. Resending request is issued in case of response timeout or incorrect data receival.
- Synchronization. All data and messages are stored in the shared memory and accessible to multiple processes. New data or messages may come in a random time. In order to prevent processes from overwriting unread data or reading "dirty" data, *mutex* primitive is used, which allows only one process to access the shared memory at a time.

C. Adaptive workload balancing

Imbalanced workload between clients posts a barrier to high efficiency. The upper plot in Fig. 6 shows the runtime for each iteration using a 3-client cluster, where each client receives the same amount of subproblems. The blue line is the runtime of the server, and the other lines correspond to the runtime of the clients. It can be seen that the runtime difference between the clients is significant. Client 3 is almost 10 seconds faster than client 1 in each iteration. Before client 1 finishes its job client 3 is idle. This leads to waste of computing resources.

There are two reasons for the imbalance. First, the computing capability of a client is determined by its hardware configuration, such as clock speed of CPUs, RAM volume, and size of cache. There are variations in the clients' hardware in a network. Second, the dimension of a subproblem depends on the length of a path. Even for the same path, the runtime of searching for an optimal solution varies against the updated Lagrangian multipliers as iteration progresses. Therefore, it is difficult to predict the time a client needs to finish its work prior to job allocation. Imbalanced workload results in long time for synchronization. Therefore, a workload balancing method must be designed to minimize the waiting time.

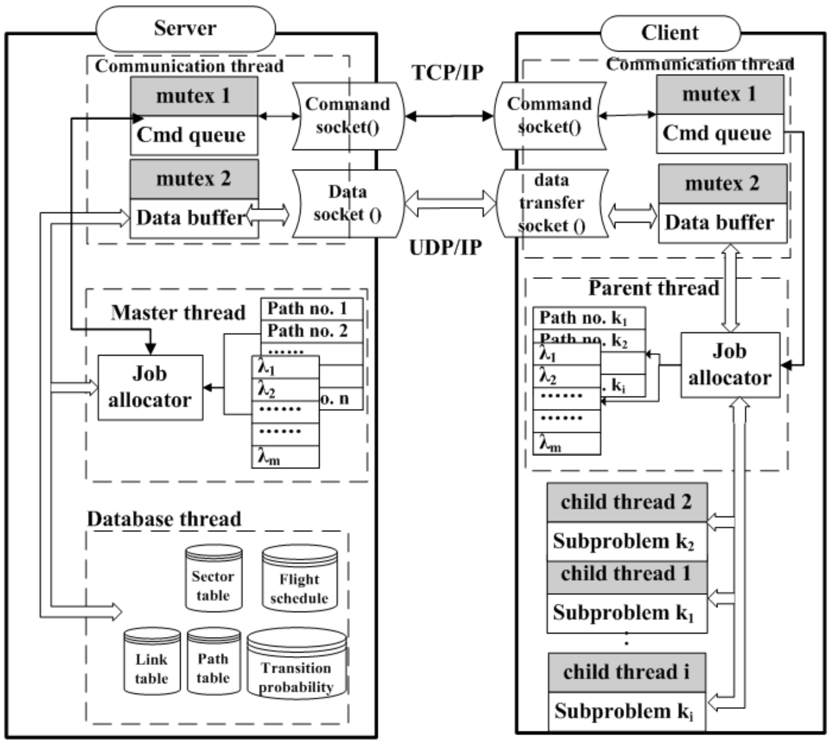


Fig. 5. Software architecture.

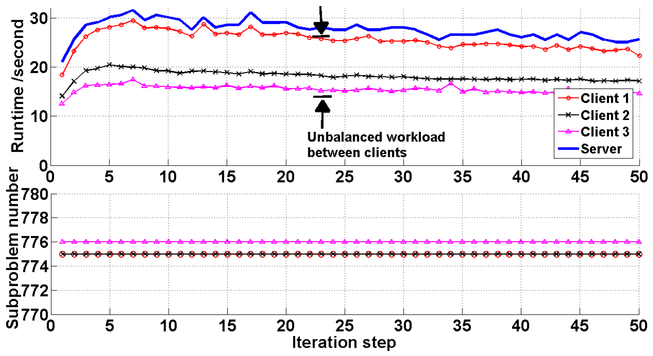


Fig. 6. 3-client without workload balancing.

Suppose there are n clients, whose computing capabilities are unknown at the beginning. The server allocates $P_l(j)$ subproblems to client l at j^{th} iteration. The j^{th} iteration on client l can be timed and the runtime which is denoted as $T_l(j)$ is sent back to the server. In the first iteration, the server evenly allocates the subproblems to each client. In the subsequent iterations, it adjusts the amount of subproblems $P(j) = [P_1(j), P_2(j), \cdots, P_n(j)]$ according to the corresponding runtime $T(j) = [T_1(j), T_2(j), \cdots, T_n(j)]$ to minimize the deviation from the mean runtime:

$$\min \sum_{l=1}^{n} |T_l(j) - \tilde{T}(j)| \tag{13}$$

where $\tilde{T}(j)$ is the average client runtime for the j^{th} iteration, calculated by $\frac{1}{n}\sum_{l=1}^{n} T_l(j)$. The deviation from the average runtime is a measurement of computing power. Runtime below the average suggests a high computing capability, and runtime above the average suggests a low computing capability. It is an intuitive idea that the server should allocate more subproblems to the high computing capability clients in order to reduce the workload of low computing capability clients, which can be achieved by the following equation:

$$P_l(j+1) = P_l(j) + \alpha \times Round(\tilde{T}(j) - T_l(j)) \tag{14}$$

where α is a coefficient that converts the computing capability into workload increment, with an unit of *paths/sec*. Operator $Rnd(\cdot)$ rounds a fraction to the nearest integer. It is assumed that the computing capability linearly correlates with the workload increment, which provides an easy heuristic to balance the workload. However, it is noteworthy that when a subproblem is shifted from one to another client the time to solve the subproblem always changes due to updated Lagrangian multipliers and different hardware. The following equation shows that the amount of subproblems is a constant. Eq. (14) just leverages the amount of subproblems between clients.

$$\begin{aligned}\sum_{l=1}^{n} P_l(j+1) &= \sum_{l=1}^{n}[P_l(j) + \alpha \times Rnd(\tilde{T}(j) - T_l(j))] \\ &= \sum_{l=1}^{n} P_l(j) + \alpha \times Rnd[\sum_{l=1}^{n}(\tilde{T}(j) - T_l(j))] \\ &= \sum_{l=1}^{n} P_l(j)\end{aligned}$$

Figure 7 shows a closed loop control method. Operator $RunOpt(\cdot)$ means runtime measurement. This is a negative feedback control system. From Eq. (14), it is clear that if client

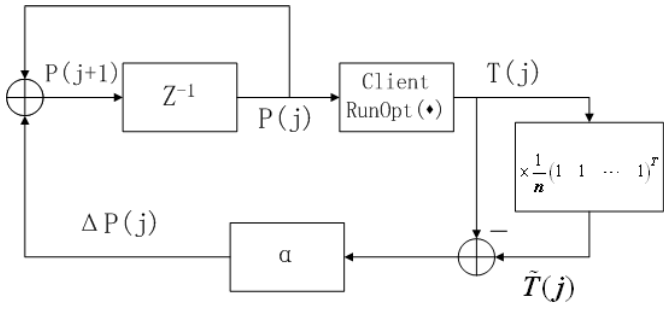


Fig. 7. Closed loop control for workload balancing.

l finishes its job more efficiently than other clients, it will receive more subproblems in the next iteration. As a result, its runtime is likely to increase. An analogy is applied to the reverse situation. The balancing process continues until the runtime of all clients is approximately homogeneous, then the amount of subproblems allocated to a client becomes stable. In such case, the workload of an individual client matches up to its computing capability. By adjusting the amount of subproblems between the clients, the time for synchronizing the clients is expected to decrease.

Eq. (14) only calculates the amount of subproblems to be reallocated. The subproblems to be reallocated are randomly determined. It is desirable to shift a set of subproblems whose aggregate runtime is close to the value $(\tilde{T}(j)-T_l(j))$. But timing the runtime of individual subproblem entails considerable overheads, therefore this method is impractical. Given that the value $(\tilde{T}(j)-T_l(j))$ is on an order of second, and the runtime difference between subproblems is generally on an order of ms, randomly choosing the subproblems is acceptable.

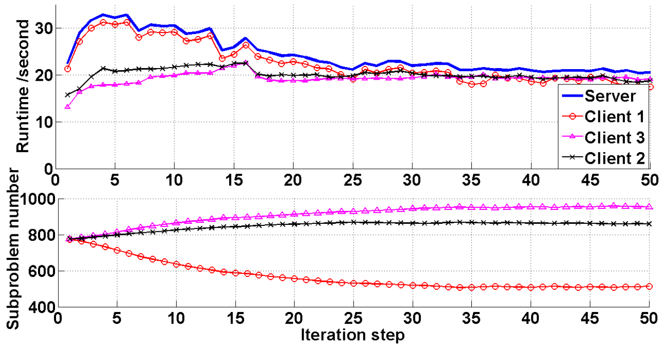


Fig. 8. 3-client with workload balancing.

Figure 8 shows the same deployment with workload balancing, where $\alpha = 3$. At the beginning of the optimization, the server evenly allocates the subproblems to each client. Since client 3 and client 2 can finish their jobs faster than client 1, the server keeps increasing the amount of subproblems allocated to them until their workload are commensurate with their computing capabilities. The runtime of the three clients are almost the same after 25 iterations, and the number of subproblems each client receives becomes stable. The workload balancing decreases the time for synchronization, thus maximizes the computing power of the platform.

The coefficient α plays an important role in the balancing procedure. A small value would make the balancing a sluggish control, whereas a great value would result in excessive tuning. It is desirable that the value of α is adaptive to the imbalance level. Generally, workload imbalance is large at the beginning, then will be diminished as the workload is tuned. A performance metric is introduced to quantify the computational efficiency of a client:

$$\beta_l(j) = \frac{P_l(j)}{T_l(j)} \tag{15}$$

$\beta_l(j)$ is the number of subproblems client l can solve within an unit of time. The standard deviation of $\beta_l(j)$ can be used as α:

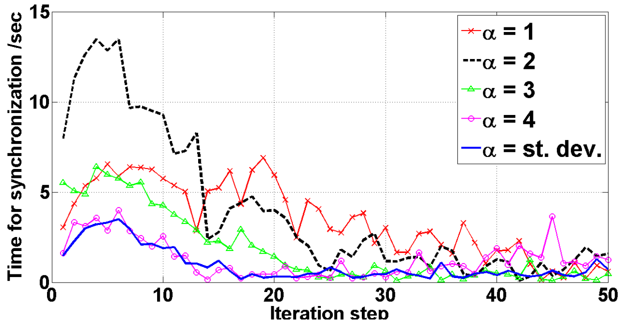
$$\alpha(j) = Round\left(\sqrt{\frac{\sum_{l=1}^{n}(\beta_l(j)-\tilde{\beta}(j))^2}{n}}\right) \tag{16}$$

where $\tilde{\beta}(j)$ is the average value for all $\beta_l(j)$. By Eq. (16), $\alpha(j)$ becomes a time varying coefficient updated in each iteration. When the workload imbalance is high, the workload increment is also high such that the workload can be tuned quickly; When the workload imbalance is low, the workload increment is also small such that the server will not overdo the balancing work.

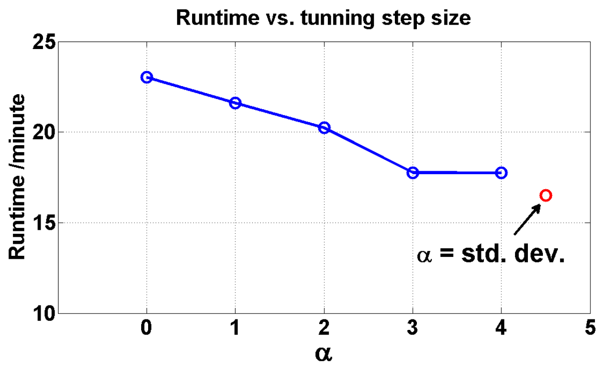
Figure 9(a) compares the transient responses using different constant values for α and the adaptive α (with the same deployment as in Fig. 6). As α increases, the system gets stable more and more quickly. For $\alpha = 4$, it takes only 15 iteration to settle down, faster than any other values. Figure 9(b) presents the runtime. It is interesting to note that $\alpha = 3$ has almost the same runtime as $\alpha = 4$, even though it takes longer to reach a stable state than $\alpha = 4$. This is due to the fact that $\alpha = 3$ experiences relatively smaller fluctuation once it gets stable, whereas $\alpha = 4$ experiences bigger fluctuation between 40 to 50 iteration. It can be predicted that the system will be more fluctuating if α is further increased due to excessive tuning. From Fig. 9(b), the system achieves the lowest runtime using the adaptive value for α. The system settles down almost as quickly as $\alpha = 4$. Besides, it manages to maintain a small fluctuation as $\alpha = 3$. Therefore, the adaptive α achieves the best performance. As can be seen in Fig. 9(b), the red circle marks the runtime achieved by using the adaptive value for α.

IV. System Validation

This section presents a 2-hour NAS-wide TFM optimization employing the proposed parallel computing framework. The traffic data are extracted from the Aircraft Situation Display to Industry (ASDI) which provides historical traffic data as well as flight plans. The peak hours traffic on March 1, 2005 is used where 2326 paths and 3054 flights were involved. Ten computers work together to perform the optimization. The computers are connected by a LAN, each with an independent IP address. Although each computer is configured with a 8-processor CPU, their computing capability is different in that the brand, generation of CPUs, and RAM size are different. Among the ten computers, one serves as the *Server*, and the rest serve as the *Clients*. The LTM is coded using C++ under Linux environment. The optimization tool used is CPLEX 12.1. The program is executed with a fixed 50 iterations to



(a) Client synchronization with different value of α.



(b) Runtime.

Fig. 9. Comparison of the overheads by different values of α.

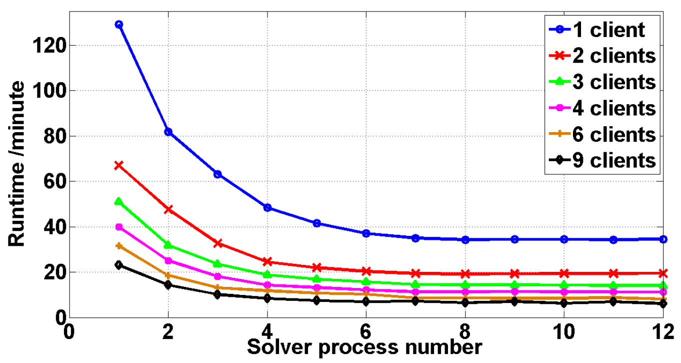


Fig. 10. Runtime of the NAS-wide TFM optimization with different client deployments.

guarantee the convergence of the algorithm. As this paper mainly focuses on parallelizing the model, only computational efficiency will be analyzed. A monolithic version of the LTM has been studied in [13]. The parallelized version yields the same optimal traffic flow as the monolithic version, but with higher efficiency.

A. Runtime and overhead analysis

To evaluate the computational efficiency, three performance metrics are defined.

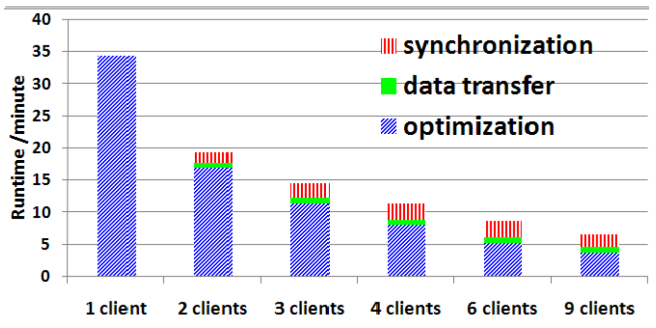
- Runtime: on the server, the runtime refers to the wall clock time that elapses from the start to the end of the optimization. Therefore, it includes the time for problem formulation, master problem update, clients synchronization, and data transfer. On the client, the runtime refers to the wall clock time for solving the subproblems and data transfer.
- Synchronization: in each iteration, the server does not update the master problem until it receives all the optimization results from the clients. The time for synchronization refers to the gap between the first feedback and the last feedback. It is included in the runtime of the server.
- Communication: communication mainly involves exchange of short messages between internal processes and data transfer between the server and the clients. Compared to the data transfer, reading/writing a short messages is quick (less than 1 ms). It is trivial to time such a process. Therefore, the time for communication refers to the total time for data transfer only in the following analysis.

Figure 10 visualizes the speedup of optimization with different configurations. As can be seen, for a fixed number of clients, the runtime decreases when more *Solver processes* are used to solve the subproblems. But such reduction is not linear. For 1-client, the speedup becomes less noticeable as the number of *Solver processes* approaches 8. One possible reason for this observation is the increasing overhead for allocating CPU time to a large number of processes. A client reaches its maximum CPU usage when the number of processes exceeds the number of processors. A similar speedup can be observed from multiple clients deployments. Compared to 1-client, 2-clients almost reduces the runtime by 50%. But further speedup is more and more difficult to achieve as more clients are added. The lowest runtime is achieved by employing 9 clients each running 12 *Solver processes*, which is nearly 6 minutes. Compared to 1-client with 1 process which takes 129 minutes, the speedup increases 21 times.

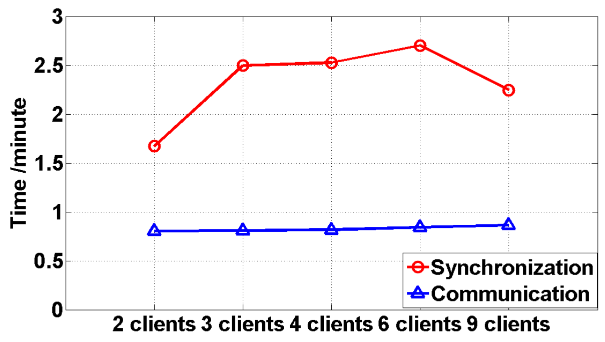
It is noteworthy that multiple clients deployments have less noticeable speedup as the number of processes increases. The speedup for 9-clients increases a little bit when the number of *Solver processes* reaches 4. This is due to the increasing overheads for clients synchronization and data transfer. Figure 11(a) shows the decomposition of runtime. When few clients are used, optimization accounts for most of the runtime. Increasing the client number will decrease the time for optimization since each client is allocated less subproblems. However, as Figure 11(b) shows, the time for synchronization and communication increase. If more clients are added, it can be predicted that the time for synchronization and communication will be more than the time for optimization itself. Then the system encounters its bottleneck.

B. Workload balancing

Increased time for synchronization is due to the heterogeneous computing capabilities of the clients. Minimizing the time for synchronization is critical to the speedup of the system. Figure 12 shows the workload allocation of the 9-client cluster. The system gets stable in about 10 iterations, then keeps fine tuning afterwards where the runtime difference between the clients is roughly less than 2 seconds. The computation of each subproblem is subject to change as the Lagrangian multipliers are updated in each iteration. It is



(a) Comparison of runtime and overheads by different client deployments (8 *Solver processes* on each client. For 1-client, the client and the server run on the same computer. There is no overhead for synchronization and communication)



(b) Average time for synchronization and communication.

Fig. 11. Overheads change as client number increases.

difficult to precisely predict how much time each client needs to finish its jobs. Therefore, as long as the runtime differences are bounded within a reasonable range, the computational efficiency of the platform is maximized.

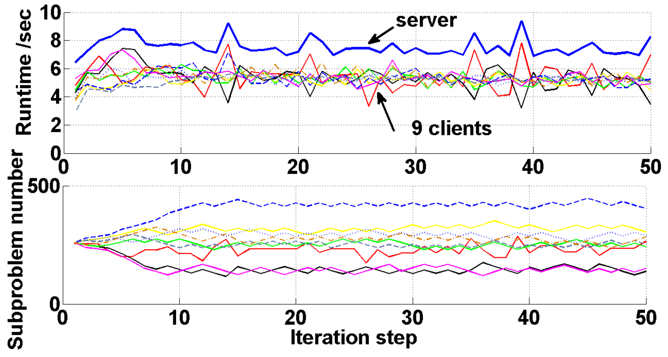


Fig. 12. Runtime and workload allocation of the 9-clients deployment.

C. Cope with increased traffic

One of advantages of this parallel computing framework is that it can easily cope with increased traffic by deploying more clients. In order to demonstrate this, the volume of traffic is increased by 50%, which is done by mixing the traffic from other time windows with current traffic. Note that the increased traffic amounts to increasing the number of independent flight paths rather than increasing the number of flights since the dimension of the LTM is related to the flight paths. There are 3419 paths corresponding to 3419 subproblems. Figure 13 compares the runtime of the increased traffic with the original traffic. The 3419-path scenario using 9 clients achieves the same computational efficiency as the 2326-path scenario using 6 clients does. The increased computation is absorbed by the increased computers. Hence, the computational efficiency is maintained.

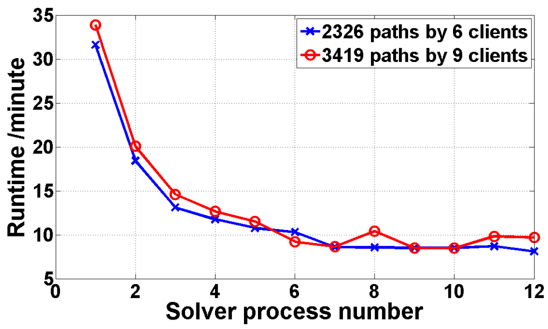


Fig. 13. Comparison of runtime with different subproblems.

D. Comparison of computational efficiency

In order to evaluate the proposed parallel computing framework, a review of historical works are summarized in Table IV. Four comparative models are designed to address the large-scale nationwide TFM problem. The runtime is decreased by developing parallel algorithm. The parallelized LTM takes longer to finish a 2-hour optimization than the parallelized CTM(L) does in that solving a MILP is more computational expensive than solving the corresponding LP relaxation. But the advantage is that there is no optimality loss due to rounding heuristic.

It is important to note that the runtime decrease does not indicate that one model is superior to the other as the decrease is a consequence of many causes. First of all, the runtime is closely related to the model used. The complexity of the BSP is proportional to the number of flight trajectories while the complexities of the CTM(L) and the LTM are proportional to the number of flight paths. Therefore, the flow-based models are more computationally efficient in modeling the NAS-wide traffic. As a trade off, the CTM(L) and the LTM lose the individual flight information. Second, the development of hardware and software package also contributes to the decrease of runtime. The most up-to-date CPU provides a solid basis for the most demanding application, while the CPLEX offers capability to solve the MILP to the common users. They make it easier to tackle the TFM problem. Third, the data sets used in model validation were different, thereby the computational complexities that the models handled vary a lot. Table I suggests that the proposed framework does provide a comparable approach for evaluation of large-scale TFM problem.

V. Conclusion

In this work, a parallel computing framework is proposed to improve the computational efficiency of the NAS-wide TFM

TABLE I
SUMMARY OF RELATED TFM OPTIMIZATION MODELS IN THE LITERATURE

Model	Year	Scope	Planning horizon	Interval /min	Solver	Hardware	Program	Runtime /min
BSP	1998	6 airports	16 hr	15	CPLEX2.1	SPARC	LP relaxation	156
Parallelized BSP	2009	NAS	2 hr	1	GLPK	8-core	LP relaxation	45
Parallelized CTM(L)	2010	NAS	2 hr	1	CPLEX12.1	8-core	LP relaxation	2.5
Parallelized LTM	2011	NAS	2 hr	1	CPLEX12.1	Multi. CPUs	MILP	6

optimization. A Link Transmission Model is used to formulate the traffic dimension depends on the flight paths identified in the NAS. The optimization is decomposed into a batch of smaller subproblems, which are solved independently on multiple clients coordinated by a server. This approach reduces the runtime by taking advantage of distributed computers to work on the optimization simultaneously.

Experiment results show that, compared to the monolithic version, a 9-client cluster is able to decrease the runtime of a 2-hour TFM optimization from over 2 hours to about 6 minutes, and it is possible to further decrease the runtime if more computers are available. But such reduction is bounded by increased overhead for synchronization and communication. An adaptive workload balancing method is developed to leverage the workload of the clients. The experiment results also show that the proposed framework is able to maintain the computational efficiency in the face of increased traffic.

In real world operations, solving the TFM problem is subject to real-time constraint. Since the NAS is such a dynamic system characterized by uncertainties like weather and tactic operations, rescheduling is necessary and frequent. Current Radar systems track the en route traffic very 1 minute, runtime less than this interval makes it possible to update the controller with"real-time" delay advisory. Apparently, current prototype does not meet this requirement, but the framework presents great potential for further runtime decrease. The improvement of efficiency presented in this paper is a big step toward this altimate goal.

REFERENCES

[1] D. Bertsimas, and S.S. Patterson, "The Air Traffic Flow Management Problem with En route Capacities," *Operations Research*, Vol. 46, No. 3, May - Jun., 1998, pp. 406-422.

[2] A.M. Bayen, R.L. Raffard, and C.J. Tomlin, "Adjoint-based control of a new Eulerian network model of air traffic flow," *IEEE Trans. on Contr. Sys. Tech.,*Vol. 14, No. 5, 2006, pp. 804-818.

[3] P.K. Menon, G.D. Sweriduk, and K.D. Bilimoria, "A New Approach for Modeling, Analysis, and Control of Air Traffic Flow," *Journal of Guidance, Navigation, Control. Dynamic*, Vol. 27, No. 5, 2004, pp. 737-744.

[4] P. K. Menon, G.D. Sweriduk, T. Lam, G. M. Diaz, and K. D. Bilimoria, "Computer-aided Eulerian air traffic flow modeling and predictive control," *AIAA Conference on Guidance, Navigation, and Control* , Vol. 29 No.1, 2006, pp. 12-19.

[5] J. Rios, K. Ross, "Massively Parallel Dantzig-Wolfe Decomposition Applied to Traffic Flow Scheduling," *AIAA Conference on Guidance, Navigation, and Control*, Chicago, Illinois, August 10-13, 2009.

[6] D. Sun, A. Clinet, and A.M. Bayen, "A Dual Decomposition Method for Sector Capacity Constrained Traffic Flow Optimization," *Transportation Research-Part B*, Vol. 45, Issue 6, pp 880-902, 2011.

[7] R.T. Rockafellar, "Augmented Lagrange Multiplier Functions and Duality in Nonconvex Programming," *SIAM Journal of Control,* Vol. 12, No. 2, May 1974.

[8] Y. Eun, I. Hwang, H. Bang, "Optimal Arrival Flight Sequencing and Scheduling Using Discrete Airborne Delays," *IEEE Trans. Intell. Transp. Sys.,* Vol. 11, No. 2, June 2010.

[9] D. Sun, A.M. Bayen, "Multicommodity Eulerian-Lagrangian Large-Capacity Cell Transmission Model for En Route Traffic," *Journal of Guidance, Control and Dynamics*. Vol. 31. No.3. May-Jun 2008, pp. 616-628.

[10] Y. Cao, D. Sun, "Performance Comparison of Two Aggregate Air Traffic Models," *AIAA Conference on Guidance, Navigation and Control*, Toronto, Canada, August 2010.

[11] Y. Ye, "An $O(n^3L)$ potential reduction algorithm for linear programming," *Math. Programming*, Vol 50, No. 2, pp. 239-258, 1991.

[12] J. Rios , K. Ross, "Parallelization of the Traffic Flow Management Problem," *AIAA Guidance, Navigation and Control Conference and Exhibit*, Honolulu, Hawaii, August 18-21, 2008.

[13] Y. Cao, D. Sun, "A Link Transmission Model for Air Traffic Flow Management," *Journal of Guidance, Control, and Dynamics,* Vol. 34, No. 5, pp. 1342-1351, 2011.

[14] "Facility Operation and Administration," February 2010, Order JO 7210.3W, U.S. Department of Transportation, FAA.

[15] D. Sun, B. Sridhar, and S. Grabbe, "Traffic Flow Management Using Aggregate Flow Models and the Development of Disaggregation Methods," *Journal of Guidance, Control and Dynamics*, Vol. 33, No. 3, May-June 2010, pp. 666-676.

[16] F.-Y. Wang, "Parallel Control and Management for Intelligent Transportation Systems: Concepts, Architectures, and Applications," *IEEE Trans. Intell. Transp. Sys.*, Vol. 11, Issue 3, 2010, pp 630-638.

[17] B. Ning, et al., "An Introduction to Parallel Control and Management for High-Speed Railway Systems," *IEEE Trans. Intell. Transp. Sys.*, Vol. 12, Issue 4, 2011, pp 1473-1483.

[18] A. H. Ghods, et al., "An Efficient Optimization Approach to Real-Time Coordinated and Integrated Freeway Traffic Control," *IEEE Trans. Intell. Transp. Sys.*, Vol. 11, Issue 4, 2010, pp 873-884.

Yi Cao is pursuing the Ph.D. degree at the School of Aeronautics and Astronautics, Purdue University. He received a bachelor degree in Instrumentation Science and Engineering in 2006, and master degree in Navigation, Guidance and Control in 2009 from Shanghai Jiao Tong University, China. His research mainly focuses on modeling, optimization, simulation, with an emphasis in air traffic flow management.

Dengfeng Sun received a bachelors degree in precision instruments and mechanology from China's Tsinghua University, a masters degree in industrial and systems engineering from the Ohio State University, and a Ph.D. degree in civil engineering from the University of California - Berkeley. Dr. Sun's research areas include control and optimization, with an emphasis on applications in air traffic flow management, dynamic airspace configuration, and studies for the Next Generation Air Transportation System (NextGen).