Particle Routing in Distributed Particle Filters for Large-scale Spatial Temporal Systems

Fan Bai, Feng Gu, Xiaolin Hu, and Song Guo

Abstract—Particle filters are important techniques to support data assimilation for large-scale spatial temporal simulation systems. Distributed particle filters improve the performance of particle filtering by distributing particles to multiple processing units (PUs). While different resampling algorithms have been developed for distributed particle filters, less research has been conducted to investigate how to route particles among the PUs after resampling in effective and efficient manners. This paper develops particle routing policies in distributed particle filters with both the centralized resampling and the distributed resampling. The developed routing policies are evaluated from the aspects of the communication cost and the data assimilation accuracy based on an application of data assimilation for large-scale wildfire spread simulations.

Index Terms—Distributed applications, Monte Carlo, routing and layout, simulation

1 INTRODUCTION

LARGE-scale spatial temporal systems, such as wildfires [1], [2], [3], road traffic [4] and floods, evolve system behaviors in both space and time. These systems typically have high dimensional state space because they cover large areas and have a large number of spatially dependent state variables. Simulation is an important tool for studying these systems. The accuracy of the simulation depends on many factors including the precision of data (e.g., wind conditions and GIS data in wildfire spread simulation) and the quality of simulation models. Both data errors and model errors will result in inaccurate simulation results. Data assimilation is a method that assimilates the real time observation data from a system to dynamically estimate the system state (usually unobservable) over time. The estimated system state can be used to set initial conditions for simulations, and thus improve simulation results. Data assimilation is generally framed as a state estimation problem using probabilistic analysis techniques. More specifically, it is the process for calculating posterior probability distributions of some state variables of interest, given prior distributions from a forecast model and data from observations. Among different data assimilation techniques, particle filters (PFs) [5] gain popularity since they are able to approximate arbitrary probability densities and have little or no assumption about the properties of the system model. They are very suitable for non-linear and/or non-Gaussian applications and show great promise in addressing a wide variety of complex problems [5], [6], [7], [8], [9], [10], [11], [12], [13].

Particle filters are sample-based methods that use Bayesian inference and statistical sampling techniques to recursively estimate the state of dynamic systems from some given observations. In applications of particle filters, the probability density function is represented by a set of particles and their associated importance weights. Sequential importance sampling and resampling (SISR) is one of the most popular particle filtering algorithms. In the SISR, two main stages include sampling and resampling. At each time step, the SISR algorithm receives a particle set representing the belief of the system state, and an observation, and then generates a number of particles representing the posterior belief. This is done by sampling from the prior distribution that represents the system transition (the simulation model). The weights of all the particles are computed according to the observation and then normalized. In the resampling step, offspring particles are drawn with probabilities proportional to the normalized weights. This procedure will be executed in an iterative manner. Particle filters have been used in data assimilation for large-scale spatial temporal systems. In previous work, we applied particle filters to data assimilation for wildfire spread simulations where ground temperature sensor data are assimilated to improve wildfire spread simulation results [3].

A major difficulty of applying particle filters to high dimensional data assimilation problems is its high computation costs due to the large number of used particles, where each particle represents a full-scale simulation to the next observation time. To improve the performance of data assimilation, distributed/parallel particle filters are needed [14]. Several distributed particle filtering algorithms have been developed and discussed in the literatures (see, e.g., [14], [15], [16], [17]). These algorithms mainly differ in how the resampling is carried out. Nevertheless, they all involve using multiple processing units (PUs) to carry out sampling of particles, and after resampling routing particles among the PUs. Particle routing is necessary because the numbers of particles on

* Fan Bai is with the Department of Computer Science, Georgia State University, Atlanta, GA 30303. E-mail: fbai1@student.gsu.edu.
* Feng Gu is with the Department of Computer Science, College of Staten Island, The City University of New York, Staten Island, NY 10314. E-mail: Feng.Gu@csi.cuny.edu.
* Xiaolin Hu is with the Department of Computer Science, Georgia State University, Atlanta, GA 30303. E-mail: xhu@gsu.edu.
* Song Guo is with the Department of Computer Science, Georgia State University, Atlanta, GA 30303. E-mail: sguo3@student.gsu.edu.
different PUs are unbalanced after resampling. Thus PUs that have surplus of particles need to route the extra particles to the PUs with shortage of particles for the next iteration of computing. As the number of PUs increases, the communication overhead increases exponentially. The unbalanced particles on PUs are caused by the fact that particles have different importance weights. As a result, PUs hosting high weighted particles generate a lot more replicates in resampling and need to route a large number of particles to others. The uneven distribution of particles’ weights is common in data assimilation using PFs for spatial temporal simulations. Fig. 1 illustrates this situation based on a run of the SISR algorithm for wildfire spread simulation [3]. The figure shows the normalized weights of the 3 “best” particles (out of total 100 particles) in each step of the 14 steps of the data assimilation. As can be seen, there is a strong uneven distribution of particles’ weights in almost every step – in some steps the 3 particles account for more than 90% of the overall weight of the 100 particles. This means in a distributed environment the PUs hosting these high weighted particles will generate a lot more replicates in resampling and need to route a large number of particles to others. Efficient particle routing is critical for reducing the communication cost in distributed/parallel PFs. This is especially true for high dimensional spatial temporal simulations because the size of each particle is large due to the high dimensional state it represents. The communication overhead depends on the level of uneven distribution of particles’ weights in each step, the state size of each particle, and the communication speed of the network. It is more significant in a distributed environment where computers are connected by networks with relatively slow speed. Therefore, particle routing is an important issue for distributed particle filters for large-scale spatial temporal systems such as wildfire spread simulation.

Particle routing deals with selecting particles on some PUs and routing them to other PUs across the network. In distributed PFs, routing particles among PUs can serve two different purposes: 1) to help the “good” particles, i.e., particles with high weights, to propagate among the PUs and thus potentially lead to better estimation results; 2) to ensure that the different PUs have the same number of particles (i.e., load balance) after resampling. While several resampling algorithms (e.g., the centralized resampling, the distributed Resampling with Proportion­al Allocation (RPA), and the distributed Resampling with Non-proportional Allocation (RNA) [14]) have been developed for distributed particle filters, less research has been conducted to investigate how to route particles among PUs after resampling in effective and efficient manners. In this paper, we study the routing policies in distributed particle filters with both the centralized resampling schema and the distributed resampling schema. In the centralized resampling schema [14], the central unit (CU) has the full knowledge of the weight distribution of all particles on different PUs. Based on this global information we propose two efficient routing policies named as minimal transfer policy and maximal balance policy in this paper. In the distributed resampling schema (more specifically, the distributed RNA), communications are constrained between neighboring PUs. This local communication schema supports a large degree of parallelism due to elimination of the centralized resampling step. However, it also results in the slow propagation of high-weighted particles, and thus reduces the convergence rate of the particles. To address this issue, we propose a hybrid particle routing approach that combines the global routing with the local routing to take advantage of both.

In this approach, we mainly use the local routing to ensure the scalability and low communication costs, and occasionally invoke the global routing to support the faster propagation of “good” particles. We evaluate and compare the different particle routing methods based on the application of data assimilation for large-scale wildfire spread simulations [18].

The different particle routing methods have different impact on data assimilation. For distributed particle filters with centralized resampling, both the minimal transfer policy and maximal balance policy can significantly reduce the communication cost. The minimal transfer policy has the best performance because it can achieve the same data assimilation results with the lowest number of particle transfers. For the hybrid approach of particle routing in distributed resampling with RNA, the maximal balance routing policy is preferred in the global routing step because it can gain the best data assimilation results with slightly more number of particles transferred compared to the minimal transfer routing policy. The contribution of this paper is to design these particle routing methods and compare their impact on both the communication cost and data assimilation accuracy based on the application of wildfire spread simulation.

The rest of this paper is organized as follows. Section 2 introduces the related work of parallel/distributed particle filters. Section 3 presents the proposed routing policies in centralized resampling, including the minimal transfer policy and the maximal balance policy. Section 4 describes the hybrid routing approach in the distributed resampling. Section 5 evaluates the proposed methods based on the wildfire spread application. Section 6 summarizes this work, discusses related issues in the proposed approaches, and points out future work.
2 RELATED WORK

Particle filters are a set of algorithms to estimate the posterior density of a system by implementing Bayesian recursion equations [5]. Particle filters are used in data assimilation of various high-dimensional systems including ocean systems, land surface systems, object tracking, and atmospheric systems. Van Leeuwan (2003) [19] applied particle filters to Agulhas Current to test the data assimilation methods because of the highly nonlinear dynamics and the availability of high-quality satellite measurement data. The dimension of the state space in this application reaches about 200,000. Zhou et al. [20] analyzed the performance of particle filters in a large-scale nonlinear land surface data assimilation example, in which a total of 684 states were considered. Mihaylova et al. [21] presented particle algorithms for filtering in group object tracking with up to 40 states and demonstrated its performance. Lingala et al. [22] adapted particle filters to one of high-dimensional chaotic systems, an atmospheric model that mimics mid-latitude atmospheric dynamics with microscopic convective processes, in which 360 dimensions were present. In spite of the aforementioned work in data assimilation, much less research has been done to use particle filters in high-dimensional systems compared to low-dimensional systems. Snyder et al. [23] pointed out the obstacles of applying particle filters in high-dimensional systems. Particle filters suffer from the "curse of dimensionality" due to collapse of particles' weights for high-dimensional systems [24]. To avoid this a large number of particles are needed, which leads to high computation costs.

To address the performance issue, different approaches have been proposed to solve problems in various applications including wireless sensor networks, traffic state tracking, robotic systems, signal processing, image processing, and target tracking. Hegyi et al. [25] presented two parallelized particle filtering algorithms to estimate the state of the freeway traffic networks based on the topological partitioning of a traffic network into subnetworks, and compared the accuracy, the computational complexity, and the communication costs of the proposed algorithm and the centralized approach. Hendhey et al. [26] used GPGPU techniques to make a parallel recursive Bayesian estimation implementation using particle filters and compared the performance with CPU implementation based on a minimal sensor network with bearings-only sensors. Mihaylova et al. [27] developed a parallelized particle filters and a parallelized Gaussian sum particle filter for online traffic management. Sheng et al. [28] proposed two novel distributed particle filters with Gaussian Mixture approximation to localize and track multiple moving targets in a wireless sensor network. Dijuric et al. [29] addressed a possible solution for improved particle filtering in high-dimensional cases by using particle filters on partitioned subspaces of the complete state space. Rosencrantz et al. [30] proposed a strictly decentralized approach in which only nearby platforms exchange information to maximize the information flow and evaluated it in a robotic system for playing the game of laser tag. Their work illustrated the scaling capability to a large team of vehicles. Other applications of parallel/distributed particle filters can be found in [31], [32], and [33]. From the mentioned work above, we can see that distributed particle filters are used in many applications, such as object tracking problems with wireless sensor networks. They have proliferated distributed sensor data as the measurement available. A lot of work mainly focused on the paradigm of the distributed particle filters and used the simple examples to evaluate the proposed methods. They used the centralized approach as the base algorithm to compare with. We also notice that two main parallel approaches were proposed. They decompose either the state space or the sample space and execute them in parallel, and the resampling stage is the main focus since the global routing is needed. In our work, we focus on the sample partitioning and its parallel execution.

Three main categories of distributed resampling algorithms exist including the centralized resampling, the distributed resampling algorithm with proportional allocation (RPA), and the distributed resampling algorithm with non-proportional allocation (RNA). Teuliere and Brun [15] parallelized particle filters using the centralized approach and applied them to Doppler-hearing tracking of maneuvering sources. A central unit (CU) was used to collect the weights from each PU to perform resampling and return replication factors to each PU. The particles were routed in a non-deterministic way. The CU simply collected surplus particles from some PUs and sent them to those that need them. Bolic et al. [14] proposed novel resampling algorithms and the architecture for the implementation of particle filters. In their work, the distributed RPA and three versions of the distributed RNA including regrouping, adaptive regrouping, and local exchange, were discussed. Different from the centralized resampling, the distributed RPA and RNA remove the central step and utilize the local weight information of PUs to decide the exchange of the particles between PUs. The distributed RPA transfers the particles according to the summed weights of PUs and the distributed RNA forms the groups of PUs and executes the resampling within the groups. Nakano and Higuchi [34] used a similar idea to group the PUs and transfer the data within them. Demirel et al. [16] introduced an adaptive distributed resampling algorithm with non-proportional allocation by dynamically adjusting the particle-exchange ratio and randomizing the process of the ring topology. Bashi et al. [35] presented three schemes for distributing the computations of generic particle filters including resampling and an optional Metropolis-Hastings step. However, how to route the particles is not discussed. In this paper, we propose detailed particle routing policies for both the centralized resampling and the distributed resampling and evaluate their impacts on data assimilation for large-scale spatial temporal systems.

3 PARTICLE ROUTING IN CENTRALIZED RESAMPLING

3.1 Overall Architecture

In the general PFs algorithm, three main steps are in-
volved including sampling, weight computation, and resampling. Among them, resampling needs information of all the particles, and thus affects the parallelization of PFs. In the centralized resampling, two types of nodes are defined, the processing unit (PU) and the central unit (CU). Sampling and weight computation are implemented on PUs since they are independent for different particles. Resampling is performed on the CU due to its sequential nature. To carry out particle routing, during the resampling stage the CU collects the weights of particles from all the PUs, performs the resampling, decides the routing information according to routing policies (described later), and then transfers particles between the CU and PUs according to the routing information. When transferring particles, PUs can send particles directly to each other. However, to simplify the overall system architecture, in our work we use the CU as a hub to collect particles from source PUs and send the collected particles to destination PUs. Note that this design choice does not affect the different particle routing policies described in this paper. The overall system architecture is illustrated in Fig. 2. In the figure, there are four PUs (PU1, PU2, PU3, and PU4) and one CU. In each particle filtering iteration, a PU carries out sampling for its particles, computes particles’ weights, and then sends the weights to the CU. After receiving all the weights, the CU normalizes the weights and performs the resampling algorithm. Consequently the CU carries out the routing procedure according to different policies (described below). According to the routing results, PUs with surplus of particles send particles to the CU, and then the CU transfers them to the PUs with shortage of particles. After the routing completes, the system evolves to the next iteration.

\[ \{1, \ldots, mn\} \mid \sum N_t^{(i,j)} = mn \]  

(1)

where \( t \) is the time step (or iteration step, used interchangeably in this paper), \( M_t^{(i,j)} \) is a selected particle and \( N_t^{(i,j)} \) is the associated number of copies for particle \( M_t^{(i,j)} \). Note that particles that are eliminated, i.e., having zero copy after resampling, are not included in \( S_t \). Given the above information, particle routing answers the following two questions: 1) How to select particles on PU with surplus of particles? 2) How to choose the destination PU for a selected particle? A routing policy will provide a solution to the above questions. An important feature of particle routing is that if multiple copies of the same particle need to be transferred across the network, only one copy of the particle plus a number indicating the duplicate number are transferred. This removal of duplicated particles reduces communication costs because the destination PU can easily make multiple copies of the received particle locally.

The routing result can be defined by a set shown in (2).

\[ R_t = \left\{ \left( M_t^{(i,j)}, p^k, S_{N_t}^{(i,j,k)} \right) : i, k \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}, \right. \]

\[ \left. S_{N_t}^{(i,j,k)} \leq N_t^{(i,j)} \mid i \neq k \right\} \]

(2)

where \( p^k \) is a destination PU for \( M_t^{(i,j)} \), and \( S_{N_t}^{(i,j,k)} \) is the number of copies for particle \( M_t^{(i,j)} \) to be sent to \( p^k \). Therefore, the routing problem can be defined as a function \( f: S \rightarrow R \), where \( S \) is the set containing the selected particles and their associated copies after resampling, and \( R \) is the set to store the routing result. Note that not all particles need to be routed to other PUs, thus the particles in \( R \) is a subset of the particles in \( S \). The routing algorithm is composed of two main steps. 1) Particle selection: this step decides how to select particles on PUs with surplus of them. 2) Destination selection: for a selected particle, a destination PU is decided to route the particle to. The algorithm is executed in an iterative manner until all the particles that need to be routed out have a destination PU assigned. Afterwards, particles are transferred according to the routing result, with duplicated copies transferred only once. The following sections present three routing algorithms based on three different particle routing policies, including the random routing policy, the minimal transfer routing policy, and the maximal balance routing policy.

3.2 Random Routing Policy

In the random routing policy, we randomly choose a particle from a PU with surplus of particles, and then select any PU with shortage of particles. Although the random routing policy may lead to large communication costs, it is still presented in this paper due to its easy implementation. We use this policy as the base to compare with other policies. Table I shows the random routing algorithm. To start the process, we first need to calculate the total numbers of copies of selected particles on each PU, and use that information to decide if a PU has surplus of particles or shortage of particles. If the total number of copies of a PU is larger than \( m \), we save its information of the select-

![Fig. 2. Overall architecture of particle routing in the centralized resampling.](image-url)
ed particles and associated number of copies in a set \( S_1 \) as shown in (3). If a PU has less than \( m \) total copies of particles, we save this PU and its needed number of particles into a set \( S_2 \) as shown in (4). Obviously \( S_1 \) is a subset of \( S_t \).

\[
S_1 = \left\{ (M_t^{(i,j)}, N_t^{(i,j)}) : i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}, N_t^{(i,j)} \in \{1, \ldots, mn\} \mid \sum N_t^{(i,j)} > m \text{ for PU with index } i \right\}.
\]

(3) \[ S_2 = \left\{ (P_t^k, R_N^k) : k \in \{1, \ldots, n\} \mid R_N^k = m - \sum N_t^{(k,j)} > 0 \text{ for PU with index } k \right\}. \]

We randomly choose a particle \( M_t^{(i,j)} \) in \( S_1 \) and a PU with index \( k \) in \( S_2 \), add the route information \( (M_t^{(i,j)}, P_t^k, 1) \) to the set \( R_t \), and then decrement \( N_t^{(i,j)}, R_N^k \), and \( \sum N_t^{(i,j)} \). If \( \sum N_t^{(i,j)} = m \) or \( R_N^k = 0 \), we remove the information for the corresponding PU from \( S_1 \) or \( S_2 \). This procedure continues until both \( S_1 \) and \( S_2 \) are empty.

### TABLE I

**RANDOM ROUTING ALGORITHM**

Main steps at time step \( t \)

1. Calculate \( \sum N_t^{(i,j)} \) for PU with index \( i, i=1, 2, \ldots, n \).
2. if \( \sum N_t^{(i,j)} > m \), save all the \( (M_t^{(i,j)}, N_t^{(i,j)}) \) for PU with the index \( i \) to the set \( S_1 \).
3. if \( \sum N_t^{(i,j)} < m \), save \( (P_t^i, R_N^i) \), where \( R_N^i = m - \sum N_t^{(i,j)} \) to the set \( S_2 \).
4. Randomly select a particle \( M_t^{(i,j)} \) in \( S_1 \).
5. Randomly select a PU with index \( k \) in \( S_2 \).
6. Append the route \( (M_t^{(i,j)}, P_t^k, 1) \) to the set \( R_t \). If \( M_t^{(i,j)} \) and \( P_t^k \) already exist in \( R_t \), increase the previous \( S_N^t \) by one but do not add \( (M_t^{(i,j)}, P_t^k, 1) \).
7. Remove information of particles on PU with the index \( i \) from \( S_1 \).
8. Remove information of PU with the index \( k \) from \( S_2 \).
9. Repeat Step 4 to 6 until both \( S_1 \) and \( S_2 \) are empty.

### 3.3 Minimal Transfer Routing Policy

While the random routing policy is easy to implement, it does not exploit the global information of particles' distribution among PUs to reduce communication costs. The minimal transfer routing policy exploits that information and aims to achieve the minimal number of particles to be sent across the network, given that replicated particles need to be sent only once between two PUs. An intuitive way to implement this is to start from selecting the particle with the most number of copies from \( S_1 \), and send them to the PU that needs the most number of particles in \( S_2 \). This reduces the overall number of transfers because the duplicated particles are transferred only once between PUs. Compared to the random routing policy, we sort the obtained set \( S_1 \) and \( S_2 \) based on the number of copies \( N_t^{(i,j)} \) and the needed number of particles \( R_N^k \) respectively in the descending order. The sorted sets are denoted as \( S_1 \) and \( S_2 \) accordingly. In each iteration, we conduct the following three steps. 1) Select the first particle (denoted as \( M_t^{(FL,F)} \)) in \( S_1 \) and its host (denoted as \( P_t^F \)) as the source PU. 2) The destination PU (denoted as \( P_t^{FK} \)) is the first PU in \( S_2 \). 3) Rout a number of copies of particle \( M_t^{(FL,F)} \) to \( P_t^{FK} \). To calculate how many copies of \( M_t^{(FL,F)} \) to be routed to the destination PU \( P_t^{FK} \), we first compare the number of copies \( (N_t^{(FL,F)}) \) of particle \( M_t^{(FL,F)} \) with the total extra number of particles \( \sum N_t^{(FL,F)} - m \) on the source PU \( P_t^F \), and select the smaller number (denoted as \( Q \)) between the two. We then compare \( Q \) with the needed number of particles \( R_N^{FK} \) on the destination PU \( P_t^{FK} \). If the former is smaller than the latter, we transfer \( Q \) copies of the \( M_t^{(FL,F)} \) to \( P_t^{FK} \), and then remove \( M_t^{(FL,F)} \) from \( S_1 \) and update the needed number of particles of \( P_t^{FK} \) by subtracting \( Q \). If the former is greater than or equal to the latter, we transfer the needed number \( (R_N^{FK}) \) of particle \( M_t^{(FL,F)} \) to \( P_t^{FK} \), update the number of copies of \( M_t^{(FL,F)} \) by subtracting \( R_N^{FK} \), and remove \( P_t^{FK} \) from \( S_2 \). In both cases, the corresponding route info is added to \( R_t \). Afterwards we re-sort the updated \( S_1 \) and \( S_2 \), and execute the same steps for the next iteration. This continues until \( S_2 \) is empty (which means all the PUs in \( S_2 \) have received the needed number of particles). The algorithm is described in Table II.

### TABLE II

**MINIMAL TRANSFER ROUTING ALGORITHM**

Main steps at time step \( t \)

1. Calculate \( \sum N_t^{(i,j)} \) for PU with index \( i, i=1, 2, \ldots, n \).
2. if \( \sum N_t^{(i,j)} > m \), save all the \( (M_t^{(i,j)}, N_t^{(i,j)}) \) for PU with the index \( i \) to the set \( S_1 \).
3. if \( \sum N_t^{(i,j)} < m \), save \( (P_t^i, R_N^i) \), where \( R_N^i = m - \sum N_t^{(i,j)} \) to the set \( S_2 \).
4. Sort the set \( S_1 \) in the descending order by \( N_t^{(i,j)} \) to \( S_1 \).
5. Sort the set \( S_2 \) in the descending order by \( R_N^i \) to \( S_2 \).
6. Select the first particle \( (M_t^{(FL,F)}) \) in \( S_1 \) to be sent and its host PU as the source PU.
7. Select the first PU \( (P_t^{FK}) \) in \( S_2 \) as the destination PU.
8. Compare \( N_t^{(FL,F)} \) with \( \sum N_t^{(FL,F)} - m \) and select the smaller number (denoted as \( Q \)) between the two.
9. Compare \( Q \) with \( R_N^{FK} \)
   a. if \( Q < R_N^{FK} \):
   
   \[ \text{PassNum} \cdot S_N^{(FL,F)k} = Q, \]
   
   \[ R_N^{FK} = R_N^{FK} - Q. \]
   
   remove \( M_t^{(FL,F)} \) from \( S_1 \).
   
   b. if \( Q \geq R_N^{FK} \):
   
   \[ \text{PassNum} \cdot S_N^{(FL,F)k} = R_N^{FK}, \]

\[ Q = R_N^{FK} - Q. \]

remove \( M_t^{(FL,F)} \) from \( S_1 \).

remove \( M_t^{(FL,F)} \) from \( S_1 \).
\[ N_{t}^{(Fl,Fj)} = N_{t}^{(Fl,Fj)} - RN_{t}^{Fl}. \]

10. Append the route information \((M_{t}^{(Fl,Fj)}, P_{t}^{Fl}, S_{t}^{(Fl,Fj)}, P_{c})\) to the set \(R_{t}\).

11. Sort the set \(S_{1}t\) in the descending order by \(N_{t}^{(i,j)}\).

12. Sort the set \(S_{2}t\) in the descending order by \(RN_{t}^{i}\).

13. Repeat Step 6 to Step 12 until \(S_{2}t\) is empty.

Fig. 3 shows an illustrative example of the minimal transfer routing policy. In the figure, there are four PUs with the indexes 01, 02, 03, and 04, and each PU has 10 particles. At time step \(t\), \(S_{1}t\) and \(S_{2}t\) have information shown on the left of the figure, where in \(S_{1}t\) each row represents a particle and associated number of copies on a PU with surplus of particles, and in \(S_{2}t\) each row is a PU with shortage of particles and the number of particles it needs. For example, PU 01 has 10 copies of particle with the index 9 \((M_{t}^{(01,9)}, 5)\) and 5 copies of particle with the index 7 \((M_{t}^{(01,7)}, 5)\), and PU 03 needs to receive 3 more particles from other PUs. According to the minimal transfer routing algorithm, \(M_{t}^{(02,5)}\) will be selected to be sent from PU 02 since it has the largest number of copies of 12. The destination PU will be the PU 04 because it needs to receive 7 particles. The source PU 02 has 5 extra copies, therefore, it will send 5 copies of \(M_{t}^{(02,5)}\) to the destination PU 04. We add the route \((M_{t}^{(02,5)}, 5, 04)\) to \(R_{t}\), which is shown in the right of the figure, where each row represents the source PU, the selected particle and number of copies, and the destination PU. After updating the related information, we continue the process until we get the final routing set \(R_{t} = \{(M_{t}^{(02,5)}, 5, 04), (M_{t}^{(01,9)}, 5, 03), (M_{t}^{(01,9)}, 2, 04)\}\. In this example, based on the routing result overall three transfers occur because the duplicated particles only need to be transferred once.

**3.4 Maximal Balance Routing Policy**

Different from the minimal transfer policy that is designed only from the communication cost point of view, the maximal balance routing policy aims to achieve the maximal balance of the particles with the high weights among all PUs after particle routing. This makes sense because the "good" particles with high weights are more likely to survive in future iterations. A balanced distribution of these particles among all PUs may reduce the need of particle routing in future iterations. The basic idea of the maximal balance policy is to select "good" particles and evenly distribute them to all PUs. Towards this goal, we need to define a criterion to decide which particles are "good". In our work, we set the criteria based on the number of copies of particles after resampling (which essentially reflect the weights of the particles). Specifically, a particle is "good" if its number of copies is greater than or equal to a threshold \(T\). Assuming the total number of "good" particles is \(G\), each PU will receive no less than \([G/n]\) (the largest integer less than or equal to \(G/n\)) and no more than \([G/n]\) (the smallest integer greater than or equal to \(G/n\)) “good” particles. To ensure that after receiving the “good” particles the total number of particles does not exceed \(m\), each PU needs to first allocate “empty” spaces for receiving the “good” particles. To support this, the maximal balance routing algorithm includes two stages. The first stage involves only the “non-good” particles. In this stage, PUs transfer particles to each other using the minimal transfer routing policy described in the previous section (in this stage each PU uses \([(mn-G)/n]\) instead of \(m\) as the desired number of particles). After the first stage, all PUs have about the same number (with plus or minus 1 difference if cannot be evenly divided) of “non-good” particles. The second stage is to distribute the “good” particles to all PUs. Specifically, we sort all the “good” particles in descending order in set \(S_{1}'\), and complete the following steps. 1) Choose the first particle in \(S_{1}'\). 2) The destination PU will be selected from all the PUs with indexes from 1 to \(n\) in turn. Every step, one copy of the first particle will be distributed to a PU by turn until all its copies are distributed. We remove this particle’s information from \(S_{1}'\), and execute the same procedure for the next particle until all the "good" are distributed.

Table III shows the maximal balance routing algorithm. We partition \(S_{1}\) into two sets \(S_{1}t\) and \(S_{2}t\). In \(S_{1}\), the weights of all the selected particles are larger than or equal to the threshold \(T\), and they are smaller than \(T\) in \(S_{2}\). Firstly we apply the minimal transfer routing policy to \(S_{2}\), and obtain the routing set \(R_{2}\). For \(S_{1}\), we evenly distribute all the copies of the particles to all the PUs, and get the routing set \(R_{1}\). The final routing set \(R_{t} = R_{1} \cup R_{2}\). If \(S_{1}\) is empty, which means no particles are considered as “good”, the maximal balance routing algorithm essentially gives the same result as the minimal transfer routing algorithm.

**TABLE III**

| \(|G/n|\) | \([G/n]\) |
|---|---|
| Maximal Balance Routing Algorithm” |  |
Main steps at time step t

1. Set a threshold \( T \).
2. Partition \( S_t \) into two sets \( S_{1t} \) (all the particles whose weights \( \geq T \)) and \( S_{2t} \) (all the particles whose weights \( < T \)).
3. Apply the minimal transfer routing policy to \( S_{2t} \) to get the routing set \( R_{2t} \). Note in this step each PU uses \( \lfloor (nm-G)/n \rfloor \) instead of \( m \) as the desired number of particles.
4. Sort \( S_{1t} \) according to the number of copies of the particles in the descending order to \( S_{1t}' \).
5. Start with the first particle in \( S_{1t}' \) and the first PU and do the following steps.
   a. Assign their copies to all the PUs one by one, and decrement its numbers of copies \( N_{i(j)} \).
   b. Append this route in \( R_{2t} \).
   c. If \( N_{i(j)} = 0 \), remove the particle from \( S_{1t}' \).
   d. Repeat the a-c process until \( S_{1t}' \) is empty.
6. The final routing \( R_t = R_{1t} \cup R_{2t} \).

Fig. 4 shows the illustrative example of the maximal balance routing policy. We use the sample example as shown in Fig. 3. Therefore, particles \( M_{t(02,5)} \) and \( M_{t(01,9)} \) are “good” particles and will be evenly distributed to all the PUs. The remaining particles will be routed by the minimal transfer routing policy and the resulting route set is \( R_{2t} = \{M_{t(03,4)} \}, 2, 02), (M_{t(03,8)} \}, 1, 04)\). Consequently, 12 copies of \( M_{t(02,5)} \) and 10 copies of \( M_{t(01,9)} \) will be transferred to all the PUs one by one. Therefore, 3 copies of \( M_{t(02,5)} \) will be assigned to each PU, and the 10 copies of \( M_{t(01,9)} \) are assigned to PUs with indexes 01, 02, 03, and 04 (2, 2, 3, and 3). Therefore, the route set is \( R_{1t} = \{M_{t(02,5)} \}, 3, 01), (M_{t(02,5)} \}, 3, 02), (M_{t(02,5)} \}, 3, 03)\), \( M_{t(02,5)} \}, 3, 04)\), \( M_{t(01,9)} \}, 2, 01), (M_{t(01,9)} \}, 2, 02), (M_{t(01,9)} \}, 3, 03)\), \( M_{t(01,9)} \}, 3, 04)\). The final routing is \( R_t = R_{1t} \cup R_{2t} \).

From the figure we see that there are totally 8 transfers between PUs since two particles are assigned to their host PUs.

4 Distributed Resampling with Local and Global Routing

The centralized resampling schema faithfully implements the particle filtering algorithm. Nevertheless, it suffers from scalability issues because it relies on a CU. To support scalable PF-based data assimilation, the distributed resampling is needed. Several distributed resampling schemas have been introduced, among which the distributed RNA uses a fully decentralized resampling schema. The main idea of the distributed RNA is no CU at all. A designer can define sub-groups among PUs and carry out full independent resampling only within the groups. In this paper, we based our work on the distributed RNA where PUs exchanges particles with local neighbors. Specifically, PUs are arranged in a ring topology and in each iteration each PU passes a subset of randomly selected particles to its neighbor in the anticlockwise order, and then carries out resampling locally. This local resampling schema supports a large degree of parallelism due to data parallelism and elimination of the centralized resampling step. However, it gives rise to a large number of iterations until full resampling is achieved. To overcome this problem, the strict local communication principle should be relaxed. Based on this idea, we propose using both local and global particle routing methods. The global particle routing is the same as in the centralized resampling algorithm described in Section 3, i.e., a CU is used to collect particles’ weights and decide how to route the particles. The goal of the global routing is to take advantage of the full knowledge of all particles’ weights to quickly and efficiently route the “good” particles to all PUs. To avoid impairing the scalability of the distributed resampling, the global routing is invoked only occasionally, e.g., once in every \( K \) steps. Table IV shows the algorithm of the distributed resampling with local and global particle routing.
on both the PU side and the CU side.

TABLE IV
ALGORITHM OF DISTRIBUTED RESAMPLING WITH LOCAL AND GLOBAL ROUTING

<table>
<thead>
<tr>
<th>Main steps at time step ( t )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PU side:</strong></td>
</tr>
<tr>
<td>For all the PUs (in parallel)</td>
</tr>
<tr>
<td>1. Give a predefined integer ( K ).</td>
</tr>
<tr>
<td>2. Run the sampling step.</td>
</tr>
<tr>
<td>3. Calculate the importance weights of particles.</td>
</tr>
<tr>
<td>4. If ( t % K = 0 ), go to step 5 (start the global resampling and routing procedure), otherwise go to step 10 (start the local routing and resampling procedure).</td>
</tr>
<tr>
<td>5. Send all weights to the CU.</td>
</tr>
<tr>
<td>6. Receive routing information from the CU.</td>
</tr>
<tr>
<td>7. If having surplus of particles, send the selected particles (based on the received routing information from CU) to the CU.</td>
</tr>
<tr>
<td>8. If having shortage of particles, receive particles from CU.</td>
</tr>
<tr>
<td>9. End.</td>
</tr>
<tr>
<td>10. Pass a subset of particles (and associated weights) to its neighbor.</td>
</tr>
<tr>
<td>11. Normalize and resampling locally.</td>
</tr>
<tr>
<td>12. End.</td>
</tr>
</tbody>
</table>

**CU side:**
1. Give the same predefined integer \( K \) as PUs.
2. If \( t \% K = 0 \), go to step 3 (activate the global resampling and routing). Otherwise skip this iteration.
3. Receive particles' weights from all PUs.
4. Normalize and resampling.
5. Compute routing information by applying the minimal transfer routing policy or the maximal balance routing policy.
6. Send the routing information to PUs.
7. Receive particles from PUs that have surplus of particles.
8. Send particles according to the routing information to the PUs that have shortage of particles.
9. End.

For the distributed resampling with local and global routing, the two different particle routing policies (the minimal transfer and the maximal balance) used by the global particle routing will have different impacts on the data assimilation results. On one hand, we expect the minimal transfer routing policy to result in less communication costs (e.g., less number of particles to be transferred). On the other hand, we expect the maximal balance routing policy to give more accurate data assimilation results. This is because the maximal balance routing policy ensures that the “good” particles are distributed to all PUs during the step of the global routing, which in turn bring positive impacts on all particles during the steps of the local routing, and thus increase the chance for the particles to better converge to the “true” state of the system. This is confirmed by the experimental results described in the next section.

5 EXPERIMENTS AND RESULTS

5.1 Experimental Design

In this paper, we evaluate the distributed resampling with the local and global routing method based on the data assimilation system of the large-scale wildfire spread simulation. The used model in this work is DEVS-FIRE, an integrated wildfire spread and suppression simulation model built on Discrete Event System Specification (DEVS) formalism [36]. More technical details of the DEVS-FIRE model can be found in [17] and [37]. In the data assimilation system, the set of fire states (fire fronts) is represented by a set of particles. The algorithm starts by initializing \( N \) particles representing the initial fire fronts when the fire is ignited. Each particle's weight is equally initiated. The algorithm goes through multiple iterations and each iteration includes sampling, weight updating, and resampling stages. At the sampling stage, all the particles run the DEVS-FIRE simulation based on the particles of last time step to generate new particles. The Gaussian graph noises are added to the generated particles and the weights of all the particles are calculated. Then we normalize the weights and use them for the resampling stage. At the resampling stage, we select the particles based on their normalized weights (particles with higher weights will generate more copies) to form a new set of particles for the next time step. More detailed description of the data assimilation for wildfire spread simulation using particle filters can be found in [3].

As many other data assimilation research, we use the widely used identical-twin experiment to evaluate the data assimilation framework system of DEVS-FIRE. The purpose of identical-twin experiments is to study the data assimilation in ideal situations and evaluate the proximity of the prediction to the true states in a controlled manner [3]. Three terms of real fire, filtered fire and simulated fire are used as in [3]. A real fire is the simulated fire spread from which the real observation data are obtained. A simulated fire refers to the simulation result based on some “erroneous” data ("erroneous" compared to the data used in the real fire). A filtered fire is the data assimilation enhanced simulation result based on the same “erroneous” data as used in the simulated fire. The goal of our experiments is to show that the different routing policies impact the performance on the distributed particle filtering.

The differences between a real fire and a simulated fire are due to the imprecise data such as wind speed, wind di-
rection, GIS data, and fuel models, used in the simulation. In our experiments, we choose to use imprecise wind conditions (wind speed and wind direction) as the “erroneous” data. Table V shows the configurations of the set of experiments. The real wind speed and direction are 8 (mph) and 180 (degrees) with random variances added every 30 minutes. The variances for the wind speeds are in the range of –2 to 2 (mph) (denoted as 8±2 in the table), and the variances for the wind direction are in the range of -20 to 20 (degrees) (denoted as 180±20 in the table). Our experiment introduces errors to the wind speeds, which are randomly generated based on the wind speed of 6 (mph) with variances added in the range of –2 to 2 (mph) and also the wind direction of 130 (degrees) with added variances in the range of ±20 (degrees) in the same time. For wind directions, the degrees indicate the angle between the north directions clockwise to the direction from where the wind comes.

**TABLE V**

<table>
<thead>
<tr>
<th>“Error” data</th>
<th>Real data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed (mph)</td>
<td>Speed (mph)</td>
</tr>
<tr>
<td></td>
<td>Direction (degrees)</td>
</tr>
<tr>
<td>6±2</td>
<td>130±20</td>
</tr>
</tbody>
</table>

In the experiment, simulations use the real-world GIS data and fuel data. The cell space dimension is 200×200 and the cell size is 20 (m). The GIS data are airborne LiDAR (Light Detection and Ranging) [38] raster-based terrain data. The fuel data were obtained by classifying a multispectral QuickBird (DigitalGlobal) image [39]. Those data were acquired from Huntsville area, Texas, during the leaf-off season in March 2004 by M7 Visual Intelligence of Houston, Texas. The ignition point is set to the point (90, 55) of the cell space for all of the simulations. The observation data (ground temperature sensor data) from the real fire are collected every 30 minutes. We use 6 PUs (every PU has 50 particles, total 300 particles) to run 6 hours’ simulation (12 steps and every step is 1,800 seconds) in all the experiments. Among these 6 PUs, one of them is functioned as a CU when completing the centralized resampling function for the global resampling step, otherwise a regular PU like others. All experiments are conducted under the supercomputer named Cheetah, which has 14 nodes, 160 computing cores, 32 CPUs, and 264 GB system memory.

### 5.2 Experimental Results

#### 5.2.1 Centralized Resampling

In this set of experiments, we conduct various experiments to show the simulation results using different routing policies including the random transfer routing policy, the minimal transfer routing policy, and the maximal balance routing policy. Fig. 5 shows the real fire spread with the real time data and the simulated fire with the imprecise wind data described in the above section. In the figures, the red and black colors represent the burning cells and burned cells respectively, and other colors refer to the unburned cells with a fuel type. From the figure we can see the real fire (as shown in Fig. 5(a)) and the simulated fire (as shown in Fig. 5(b)) are obviously different regarding the spread direction, and burned areas. After assimilating the real time data in the simulation, we expect the improved fire spread estimation. Fig. 6 displays the real fire, the simulated fire, and the filtered fire by assimilating the real data into wildfire spread simulation using the random routing policy, the minimal transfer routing policy, and the maximal balance routing transfer policy respectively. In the figures, all the filtered fires (display in red) are close to the real fire (display in blue) although we run the data assimilation simulations with the error data. Compared to the simulated fires (display in black), all the simulation results are greatly improved.
Fig. 6. Comparisons of real fire, simulated fire, and filtered fire using different routing policies. (a) Random routing policy. (b) Minimal transfer routing policy. (c) Maximal balance routing policy.

To quantitatively examine the results, we choose the symmetric set difference as the metric to measure the similarity of the fires. In mathematics, the symmetric set difference of two sets is the set of elements in either set, but not in both. We use it to compare two fire fronts, which is the number of cells inside one of the fire front shapes, but not in both. The smaller the symmetric set difference, the more similar the two fire fronts are (the symmetric set difference of two same fire fronts is 0). Fig. 7 shows the symmetric set differences of the simulated fire (compared to the real fire) and that of the filtered fire (compare to the real fire) using three routing policies including the random routing policy, the minimal transfer routing policy, and the maximal balance routing policy. In the figure, the values of the filtered fires are the average of 6 independent runs. The horizontal axis represents the time step, and the vertical axis represents the symmetric set difference value in terms of the number of cells. From the figures, it can be seen that the symmetric set differences of the filtered fires are smaller than those of the simulated fires after step 5. With the increase of the time step (i.e., when more sensor data are assimilated), the difference between the simulated fire and the filtered fire becomes more and more notable. At step 12, the symmetric set difference of the filtered fire is more than half of the symmetric set difference of the simulated fire. Also, the simulation results which use three different routing policies get the similar accurate results. From Fig. 6 and Fig. 7, we conclude that the data assimilation using three different routing policies including the random routing policy, the minimal transfer routing policy, and the maximal balance routing policy all significantly improve the simulation results. There is little difference among the three policies from the simulation results point of view. This is expected because in the centralized resampling all particles are resampled in each step. The three policies differ only in how particles are routed after resampling and thus only impact the communication cost but have little impact on the data assimilation results.

To compare the communication cost of the three policies, we use the number of transferred particles since it is proportional to the communication cost if the network speed is fixed and the sizes of particles are roughly the same. It counts the number of transferred particles between any two PUs during the data assimilation process. Fig. 8 shows the number of transferred particles in every step for the random routing policy, the minimal transfer routing policy, and the maximal balance routing policy. Fig. 9 shows the total number of transferred particles of the total 12 steps for the random routing policy, the minimal transfer routing policy, and the maximal balance routing policy. From the figures we know that both the minimal transfer routing policy and the maximal balance routing policy significantly reduce the number of transferred particle and the minimal transfer routing policy has the lowest number of transferred particles.

Fig. 7. Symmetric set differences for simulated fire and filtered fire with three different routing policies.

Fig. 8. Numbers of transferred particles for the random routing policy, the minimal transfer routing policy, and the maximal balance routing policy.

Fig. 9. Total number of transferred particles of the total 12 steps for the random routing policy, the minimal transfer routing policy, and the maximal balance routing policy.
5.2.2 Distributed Resampling

In this section, we do the same experiment using the distributed RNA, the distributed RNA with the minimal transfer routing policy, and the distributed RNA with the maximal balance routing policy. The last two are hybrid particle routing approaches because they combine both local routing and global routing. For the distributed RNA policy, each PU passes 10 particles to its neighbor in the anticlockwise order. For the latter two policies, we call the centralized resampling (the minimal transfer routing policy or the maximal balance routing policy) every 4 steps, and remove the duplicate particles. However, we don't do this in the distributed RNA policy, so its number of transferred particles is 60 for 6 PUs in each step.

Fig. 10 displays the number of transferred particles of the three policies of the distributed RNA, the distributed RNA with the minimal transfer routing policy, and the distributed RNA with the maximal balance routing policy for step 4, step 8, and step 12 respectively. Fig. 11 shows the total number of transferred particles of the three policies of the distributed RNA, the distributed RNA with the minimal transfer routing policy, and the distributed RNA with the maximal balance routing policy. Note that all the values are the average of 6 independent runs. The numbers of transferred particles are the same for all the steps except step 4, step 8, and step 12. This is because we apply the minimal transfer routing policy or the maximal balance routing policy ever four steps, and the distributed RNA is used for other steps. For the steps where the minimal transfer routing policy or the maximal balance routing policy is applied, the numbers of the transferred particles are greatly decreased, and they are reduced more by the distributed RNA with the minimal transfer routing policy. However, there is less obvious difference between the distributed RNA with the minimal transfer routing policy and the distributed RNA with the maximal balance routing policy regarding the total number of transferred particles.

Fig. 12 shows the symmetric set differences between the real fire and the filtered fire using the centralized resampling with the minimal transfer routing policy, the filtered fire using the distributed RNA, the filtered fire using the distributed RNA with the minimal transfer routing policy, and the filtered fire using the distributed RNA with the maximal balance routing policy respectively from step 7 to 12. Because the fire is small in the earlier steps and thus the differ-
ence is small, we skip the early steps and zoom in the later steps in order to better show the results. From the figure we can see that although all of them have much less symmetric set differences than the simulated fire mentioned above, the distributed RNA has the worst results since it is a purely distributed resampling and suffers from the local resampling. Fig. 13 displays the symmetric set differences for the distributed RNA, the distributed RNA with the minimal transfer routing policy, and the distributed RNA with the maximal balance routing policy at time step 8 and 12 respectively. At these two steps, both the distributed RNA with the minimal transfer routing policy and the distributed RNA with the maximal balance routing policy have smaller symmetric set differences than the distributed RNA, and the distributed RNA with the maximal balance routing policy has better results than the distributed RNA and the distributed RNA with the minimal transfer routing policy. This is because the maximal balance routing policy evenly distributes the copies of particles with high weights to all the PUs during the routing process. Thus the distributed RNA with the maximal balance routing policy has the best simulation results among the three distributed resampling methods even though it has slightly more total number of transferred particles compared to the distributed RNA with the minimal transfer routing policy. To better characterize and compare the accuracy of the simulation results, for each method we also calculate the root-mean-square error (RMSE) of the symmetric set differences at step 12 based on the 6 experiment runs. The RMSE for the centralized resampling is 3,877.54; the RMSE for the distributed RNA is 4,525.77; the RMSE for the distributed RNA with the minimal transfer routing policy is 4,364.63, and the RMSE for the distributed RNA with the maximal balance routing policy is 4,316.83. These measurements confirm that the centralized resampling leads to more accurate result compared to the distributed resampling. Among the distributed resampling methods, the distributed RNA with the maximal balance routing policy has the best accuracy.

![Fig. 12. Symmetric set differences for the filtered fire with the centralized resampling using the minimal transfer routing policy, and the filtered fire with the distributed RNA, the filtered fire with the distributed RNA with the minimal transfer routing policy, and the filtered fire with the distributed RNA with the maximal balance routing policy.](image1)

![Fig. 13. Symmetric set differences for filtered fire using the distributed RNA, filtered fire using the distributed RNA with the minimal transfer routing policy, and the filtered fire using the distributed RNA with the maximal balance routing policy at time step 8 and 12.](image2)

The frequency of applying the centralized resampling to the decentralized sampling is an important factor. It is the tradeoff between the communication cost and the precision of the simulation results. We duplicate the experiments by calling the centralized resampling every 2 step and 5 steps. Fig. 14 displays the symmetric set differences of the filtered fire using the distributed RNA and the distributed RNA with the maximal balance routing policy for every 2 steps, 4 steps and 5 steps at time step 12. From the figure we know that the distributed RNA with the maximal balance routing policy every 2 steps has the smallest symmetric set difference and thus has the best simulation results. This is because the global routing is applied more often. The distributed RNA with the maximal balance routing policy every 5 steps has the worst results since it only does the global routing twice.

![Fig. 14. Symmetric set differences of the filtered fire with the distributed RNA, and the distributed RNA with the maximal balance routing policy.](image3)
policy every 2 steps, 4 steps and 5 steps at time step 12.

Fig. 15 displays the total number of transferred particles for the distributed RNA with the minimal transfer routing policy and the distributed RNA with the maximal balance routing policy for every 2, 4, and 5 steps. From the figure we can see that if more centralized resampling steps are added, more numbers of transferred are needed. However, for the same number of added centralized resampling steps, the distributed RNA with the minimal transfer routing policy has less number of state transfers. This is consistent with the previous observations.

Fig. 15. Total numbers of transferred particles of the distributed RNA with the minimal transfer routing policy and the distributed RNA with the maximal balance routing policy for every 2, 4, and 5 steps at time step 12.

6 DISCUSSIONS AND CONCLUSIONS

In this work, we propose two centralized particle routing policies named as the minimal transfer routing policy and the maximal balance routing policy and show their impacts on distributed particle filters with the centralized resampling. We also propose a hybrid approach that uses both local and global particle routing in the distributed resampling with nonproportional allocation (RNA). We show how the minimal transfer routing policy and the maximal balance routing policy can be used in the global routing step and their impacts on performance and accuracy of particle filtering. We evaluate the proposed methods based on data assimilation of a large-scale wildfire spread simulation. Experimental results show that the minimal transfer routing policy is the best choice for the centralized resampling because it can achieve the same data assimilation results with the lowest number of state transfers compared to the random routing policy and the maximal balance routing policy. For the hybrid approach of particle routing in distributed resampling with RNA, the maximal balance routing policy is preferred in the global routing step because it can gain the best data assimilation results with slightly more number of state transfers compared to the minimal transfer routing policy. These results provide a guideline for choosing the different particle routing policies for other applications. Note that in the experiments, we used a relatively small number of PUs (6). When the number of PUs increases, the differences between the proposed methods could be more significant, especially in a distributed environment.

In this paper, we implement both the minimal transfer routing policy and maximal balance routing policy in an intuitive manner without formally proving that the algorithms will always guarantee best results. A formal analysis of these algorithms will be a task for our future work. The experimental results shown in this paper are based on a specific application of data assimilation of wildfire spread simulation. In general, the performance of the different particle routing policies are dependent on the distribution of particles’ weights among PUs. If all PUs has balanced distribution of particles’ weights, the different policies will not lead to much different results because there is little need to transfer particles between PUs. On the other hand, if all the high weighted particles are concentrated on a single PU, the different policies will not lead to much different results either because they all result in transferring particles from the dominant PU to others. Systematically and formally analyzing in what conditions the different routing policies perform the best is another task that we plan to carry out in future work.

ACKNOWLEDGMENT

This work was supported in part by the US Natural Science Foundation under Grant CNS-0841170 and CNS-0941432 and 1356977. The authors thank the anonymous reviewers for their valuable comments that helped improve the quality of this paper.

REFERENCES

[10] F. Dellaert, “A Sample of Monte Carlo Methods in Robotics and


Fan Bai received his B.S. in computer science and technology at XiDian University in 2007 and M.S. in computer science at Georgia State University in 2011 respectively. He is currently a Ph.D. candidate of computer science at Georgia State University. His research interests include modeling and simulation, discrete event system specification, model-based development, and software engineering.

Feng Gu received his B.S. degree in mechanical engineering from China University of Mining and Technology in 1998 and M.S. degree in information systems from Beijing Institute of Machinery in 2003. He received his M.S. and Ph.D. in computer science from Georgia State University in 2009 and 2011 respectively. He was an assistant professor of computer science at Voorhees College from 2010 to 2013. He is currently an assistant professor in the Department of Computer Science at College of Staten Island, The City University of New York, and the doctoral faculty member of Graduate Center of The City University of New York. He is the recipient of the Natural Science Foundation Research Initiation Award. His research interests include modeling and simulation, complex systems, high performance computing, and bioinformatics.

Xiaolin Hu received his Ph.D. degree from the University of Arizona in 2004. He is currently an associate professor of computer science at Georgia State University and the director of the Systems Integrated Modeling and Simulation (SIMS) Lab. His research interests include modeling and simulation, agent and multi-agent system, and complex systems science. He is a National Science Foundation CAREER Award recipient.
Song Guo received his B.S. in computer science and technology at Beijing University of Chemical Technology in 2005. He received his M.S. and Ph.D. in computer science from Georgia State University in 2009 and 2012 respectively. His research interests include modeling and simulation, parallel and distributed computing, web service and service-oriented architecture.