J. Parallel Distrib. Comput. 93–94 (2016) 102–119

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Efficient parallel simulation of spatially-explicit agent-based epidemiological models

Dhananjai M. Rao

CSE Department, Miami University, Oxford, OH, 45056, USA

HIGHLIGHTS

- Modeling methodology for spatially-explicit epidemic models.
- Model with high risk migratory waterfowl involved in global spread of avian influenza.
- Design of a parallel and distributed modeling, simulation, and analysis environment.
- Proxy-based approach for logical migration of agents during simulation to reflect migratory characteristics.
- Novel ghosting of agents to handle boundary cases that arise during migration to ensure scalable and efficient parallel simulation.

ARTICLE INFO

Article history:
Received 9 March 2015
Received in revised form
6 April 2016
Accepted 7 April 2016
Available online 19 April 2016

Keywords:
Epidemiology
Avian influenza
Agent-based model
Spatially-explicit model
Parallel Discrete Event Simulation (PDES)
Time Warp
Logical process migration
Ghosting

ABSTRACT

Agent-based approaches enable simulation driven analysis and discovery of system-level properties using descriptive models of known behaviors of the entities constituting the system. Accordingly, a spatially-explicit agent-based ecological modeling, parallel simulation, and analysis environment called SEARUMS has been developed. However, the conservatively synchronized parallel simulation infrastructure of SEARUMS did not scale effectively. Furthermore, the initial multithreaded shared-memory design prevented utilization of resources on multiple compute nodes of a distributed memory cluster.

Consequently, the simulation infrastructure of SEARUMS was redesigned to operate as a Time Warp synchronized parallel and distributed discrete event simulation (PDES) on modern distributed-memory supercomputing platforms. The new PDES environment is called SEARUMS++. The spatially-explicit nature of the models posed several challenges in achieving scalable and efficient PDES, necessitating new approaches in SEARUMS++ for: ① modeling spatial interactions and initial partitioning of agents, ② logical migration of an agent during simulation using proxy agents to reflect migratory characteristics, and ③ ghosting of agents using multiple proxy agents to handle boundary cases that occur during logical migration of agents.

This article presents our optimization efforts involving new methods to address aforementioned challenges. The design of SEARUMS++ and experimental evaluation of various alternatives that were explored to achieve scalable and efficient PDES are also discussed. Our experiments indicate that SEARUMS++ provides 200% performance improvement and maintains scalability to a larger number of processors, thus enabling efficient parallel simulation of spatially-explicit agent-based epidemiological models.

1. Introduction

Humanity faces a multitude of global socioeconomic challenges due to recurring epidemics and punctuated pandemics of Highly Pathogenic Avian Influenza (HPAI) viruses, specifically of H5N1 and H7N9 serotypes [28,37,16,40]. Infected migrating waterfowl, in which the HPAI is endemic, are the primary vectors causing intercontinental spread of the disease [20,28]. The virus rapidly spreads from waterfowl to poultry, swine, and humans through contaminated water, feed, and surfaces as summarized in Fig. 1. The 2009 H1N1 pandemic was caused due to a novel viral strain involving a triple reassortment of avian, swine, and human serotypes [37]. Several recent investigations have established that with just 1 mutation the H5N1 virus gains the ability to be readily transmitted between humans and thereby significantly worsening the impacts of epidemics [16,42]. Consequently, there is heightened urgency to

E-mail address: raodm@miamiOH.edu.

http://dx.doi.org/10.1016/j.jpdc.2016.04.004
0743-7315/

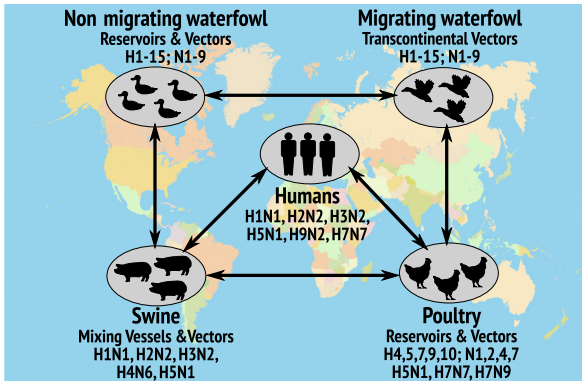


Fig. 1. Overview of global ecology of avian influenza.

shift the focus of investigations from studying possibilities to analyzing probabilities of outbreaks and to *proactively* mitigate or even preempt emergent pandemics [37]—a contrast to the delayed, *reactive* responses to the 2009 pandemic [37] and its adverse consequences as discussed in the report by the executive office of the president of the United States [27].

Importance of modeling & simulation: Proactive approaches for mitigation of emergent epidemics rely on computational methods that integrate computer science and classical epidemiology [11,40,41]. Computational epidemiology enables comprehensive understanding of disease ecology and facilitates proactive design and administration of prophylactic strategies. It is fundamentally based on Modeling and Simulation (M&S) with added integration of statistics, bioinformatics, and geography for various analysis. Recently, a variety of computational approaches have been proposed to analyze ecological, epidemiological, and socioeconomic risks of HPAI [28,37,11,40].

Our computational epidemiology approach: In contrast to other investigations, our approach aims to enable comprehensive, global analysis by explicitly modeling migratory patterns of waterfowl, the primary transcontinental vectors for avian influenza (see Fig. 1). Since, the degree and extent of the influence of migratory patterns on disease ecology is poorly understood [32,12, 44], we propose the use of detailed temporo-geospatial models of migratory patterns generated from surveillance data [33] to elicit global disease characteristics [32]. Fig. 2 illustrates such a model viewed in SEARUMS, the M&S environment used and enhanced in this study (see Section 4 for details). Agents, represented as circles in Fig. 2, interact with each other when they overlap. Overlap between agents is detected using a separate "spatial" agent that tracks their geospatial locations. Section 4 summarizes results from our earlier publications [35,33,32] by discussing our model [33], its validation [32], and its successful application to analyze epidemiological, ecological, and socioeconomic impacts of HPAI [35,32].

1.1. Motivation for parallel & distributed simulation

The advantages of agent-based models (ABMs), including our spatially-explicit ABMs of migratory waterfowl, are realized at the cost of significantly higher simulation execution time because ABMs are computationally demanding. Consequently, the Java-based simulation kernel of SEARUMSwas initially designed to operate using multiple threads on shared-memory multi-core and symmetric multiprocessing (SMP) systems using conservative synchronization method [34,17]. The multithreaded simulation kernel used a centralized event queue with minimal locking for scheduling and processing timestamped events in correct causal order. Multiple threads process concurrent events scheduled at a given simulation time. A more detailed description of the simulation kernel is available in the literature [34].

The multithreaded Java-based simulation back-end of SEARUMS performed well for many of our test models. The chart in Fig. 3 plots the expected versus observed runtime for a Synthetic Test Model STM, with 360 agents created by replicating 30 agents 12 times. The replicated agents have identical behaviors and schedule events at exactly the same virtual time, thereby ensuring at least 12 concurrent events are scheduled at each time step in the simulation. The motivation for 12 replications stems from the CPU architecture of the compute node used to conduct experiments. The experiments were conducted in headless mode (to avoid all GUI overheads) on a dedicated compute node with two hex-core (total 12 cores) Intel® Xeon™ X5650 CPUs @ 2.67 GHz. The experiments were conducted using the `-XX:+UseNUMA` and `-XX:+UseParallelGC` JVM flags designed to mitigate the performance impacts on the

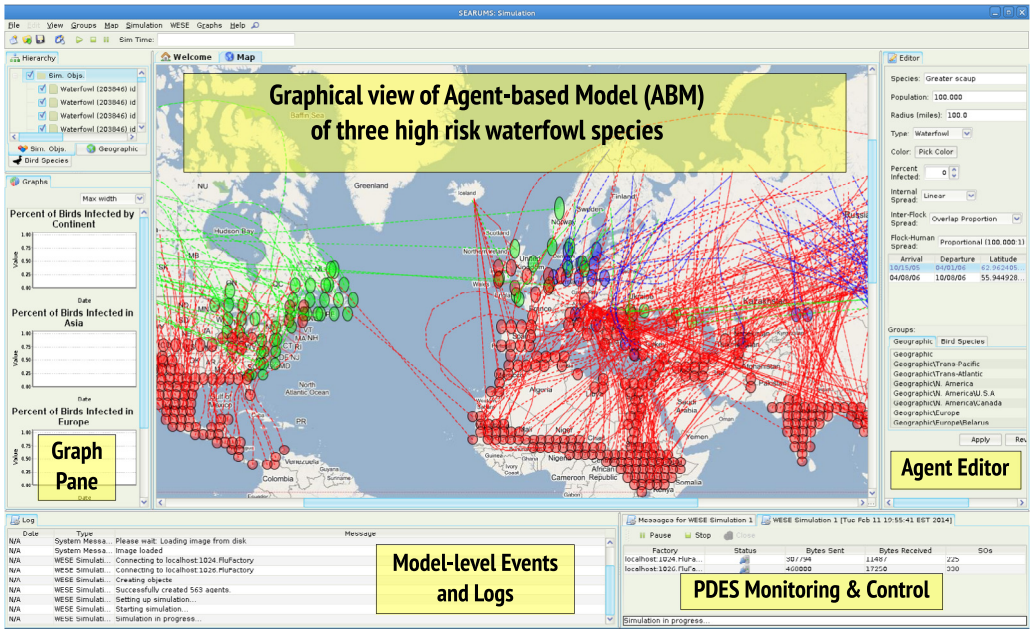


Fig. 2. Screenshot of an Agent-based Model (ABM) viewed in SEARUMS, the M&S environment used in this study (see Section 4 for details).

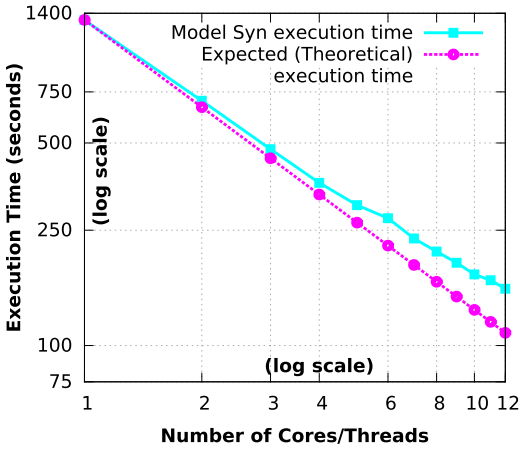


Fig. 3. Scalability testing of Java-based, multithreaded simulation kernel using a Synthetic Test Model (STM) with 360 agents.

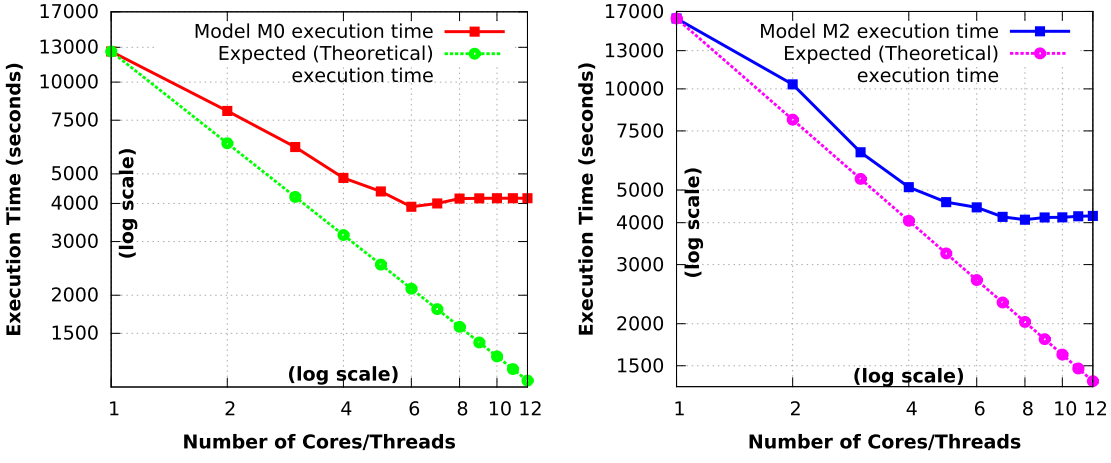
NUMA architecture of the Xeon™ X5650 CPU. The chart in Fig. 3 shows that the simulator scales well, with some degradation in speedup around 10 cores. The degradation in speedup at and above 10 cores is expected because some of the CPU time is occupied by the JVM's garbage collection threads and by OS processes.

However, for real-world models, the simulation kernel did not scale as expected. Fig. 4 presents a comparison between theoretical and observed simulation execution times. The experiments were conducted using two models, namely M1 (with 2628 agents) and M2 (with 6455 agents). As shown in Fig. 4(a), given a single core runtime of 12 600.17 s for M1, the observed speedup with 6 cores was just $12\,600.17 \div 3900.23 = 3.23$, when compared to speedup of 5.25 for the synthetic model (see Fig. 3). Furthermore, as illustrated by the theoretical versus observed curves in Fig. 4(a), the scalability for M1 tappers off at six cores. Model M2 (see Fig. 4(b)) had similar scalability characteristics, with improvements tapering off at about 8 cores (speedup of ∼3.9), even though M2 was a larger model with higher concurrency.

Shortcomings of initial conservative approach: Detailed analysis of the virtual timestamped discrete events in model M2 was conducted to assess availability and utilization of concurrency in the model. Fig. 5 shows a plot of available concurrency in the model versus the used concurrency. The concurrency data in Fig. 5 was measured through instrumentation of the simulator in the following manner. For an agent a, let $lvt(a)$ (at initialization $lvt(a) = 0$) be the virtual time when agent a executed its chronologically previous event. For an event e, let and $t_{recv}(e)$ be the virtual time when agent a must process event e. Moreover, in our model the virtual time of events (*i.e.*, t_{recv}) is used such that just t_{recv} is sufficient to yield a total ordering of events for each agent without requiring any additional information (such as process IDs) to break ties. In a causally correct conservative simulation when an agent a receives an event, $t_{recv}(e) > lvt(a)$. Moreover, event e is processed by agent a only when the global clock advances to $t_{recv}(e)$. However, in the general case, if agent a does not receive any other event between $lvt(a)$ and $t_{recv}(e)$, then it can process e at $lvt(a)$ without violating causality, rather than waiting for the global clock to advance to $t_{recv}(e)$. Accordingly, in the general case, the earliest simulation time agent a can process an event e is $lvt(a)$, where $lvt(a)$ is the virtual time it processed its chronologically preceding set of events. This relationship has been used to track concurrency in a single threaded simulation.

Specifically, just before an event e is processed by an agent at virtual time $t_{recv}(e)$, a counter corresponding to $lvt(a)$ in a global table is incremented to track the number of events that could have processed at time $lvt(e)$ (note $lvt(e) < t_{recv}(e)$) without violating causality. This value is a measure of maximum available concurrency in the model at each time step and is shown by the light green region in Fig. 5. The dark green solid curve is fitted to the maximum available concurrency observations to illustrate general trend. In addition, the actual number of events that were processed by agents at time $t_{recv}(e)$ is tracked in another global table. This value is a measure of concurrency actually used by the conservative simulator and is shown by the cyan region in Fig. 5 with the solid blue curve being fitted to the data to show trends.

Fig. 5 indicates that there is significant potential parallelism available in the model. In conjunction with the graphs in Fig. 4, it is inferred that the multithreaded Java kernel is unable to effectively extract and exploit the latent parallelism available in the model. Further investigations revealed that the root cause for this bottleneck was the stochastic nature of real-world models which have variances in timings. In other words, unlike the synthetic model STM (see Fig. 3) that scheduled 12 events at time t, the actual models schedule events at time $t \pm \Delta t$, where Δt is stochastic variance necessary to characterize probabilistic behavior of birds in nature. The variance reduces the number of events scheduled at exactly the same virtual time t. In such a scenario, several threads remain idle waiting for previous events to be processed to avoid causal violations. Unfortunately, such scenarios occur frequently in the model, thus negatively impacting the overall scalability and performance of the simulator. Moreover, the supercomputing cluster available for our research had a



(a) Model M1.

(b) Model M2.

Fig. 4. Performance of Java-based, multithreaded simulation kernel for models M1 and M2.

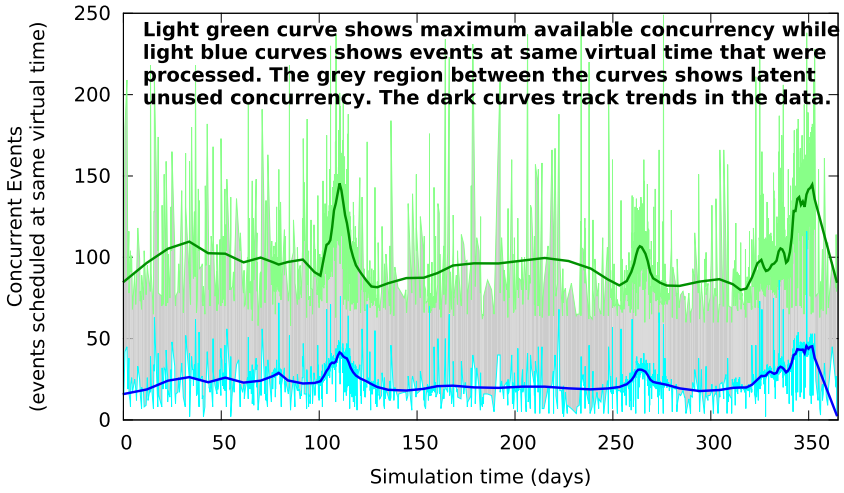


Fig. 5. Chart illustrating the concurrency that is unused by the conservatively synchronized multithreaded Java simulation kernel. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

distributed memory architecture and the shared-memory design could not effectively utilize the computational infrastructure beyond a single node, as summarized in the following subsection.

1.2. Overview of proposed research and this article

Endeavoring to address the shortcomings of our conservatively synchronized shared-memory simulation kernel, we have redesigned the simulation infrastructure of SEARUMSto operate as an optimistically synchronized parallel and distributed simulation. Optimistic synchronization, specifically Time Warp [17] was chosen as it has shown to maximize parallelism and scale effectively for spatially-explicit models and epidemiological simulations [24,7]. As detailed in Section 5, the simulation infrastructure was redesigned in C++ using a general purpose, Time Warp synchronized simulation framework called WESE [30]. The chart in Fig. 6 summarizes the runtime for model `M2` using our revised simulation infrastructure. As illustrated by the chart in Fig. 6, the revised approach provides improved scalability when compared to the initial conservative version shown in Fig. 4(b). Section 8 provides additional details on performance analysis of the revised infrastructure along with discussion on model-related factors that influence overall scalability in the revised system.

The earlier shared-memory approach used a unified spatial agent (called `UnifiedEcoArea`) for detecting overlap between agents and initiating various interactions [34]. The `UnifiedEcoArea` approach had to be redesigned into multiple independent spatial agents (called `SplitEcoAreas`) to operate efficiently on distributed memory platforms as discussed in Section 5. However, as discussed in Section 5, the redesigned `SplitEcoArea` approach did not readily scale as the number of *mobile* agents, representing migrating waterfowl flocks, was increased.

The root cause of the issue was that the mobile agents do not *physically* move (the C++ class instantiated for an agent is fixed on a compute node), but only move *logically* by changing latitude and longitude values in their state. Nevertheless, since all interactions are event driven, the logical movement of agents causes large number of Inter-Process Messages (IPMs) that flow over the interconnects resulting in increased synchronization overheads and degraded performance. In order to address this bottleneck, as detailed in Section 5.4, a Single Active Proxy (SAP) approach is proposed. In SAP, as agents logically migrate, *proxy* agents on a different compute nodes are suitable activated and deactivated to embody migratory patterns and to minimize IPMs.

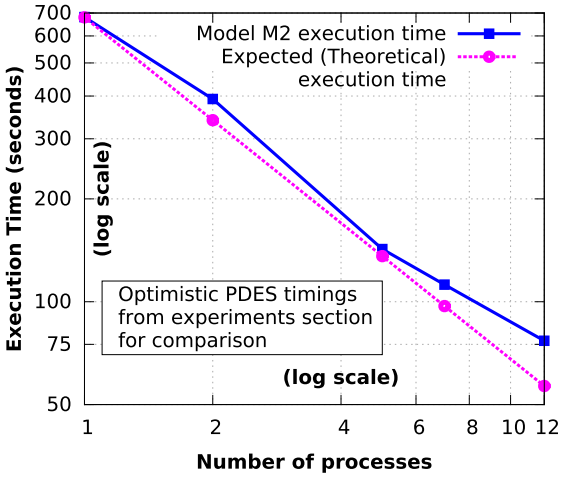


Fig. 6. Performance of the revised Time Warp synchronized simulation infrastructure for model `M2`. Additional experimental data along discussion on model-related factors that influence overall scalability of the revised infrastructure are discussed in Section 8.

However, as detailed in Section 5.5, the SAP approach still experienced performance issues at boundary cases when an agent's neighborhood spans two or more partitions. Consequently, a Multiple Active Proxy (MAP) approach is proposed to address three different boundary cases that arose in SAP approach. The MAP approach extends the SAP approach by permitting multiple proxies to be active at the boundary cases when an agent spans two or more partitions to minimize IPMs. Section 6 presents the MAP approach followed by Section 7 that contrasts the proposed methods with related research. Section 8 discusses results from various experiments conducted to assess the proposed methods. Section 9 concludes the paper by summarizing the outcomes and inferences drawn from this investigation.

2. Background on epidemiological models

Modeling and Simulation (M&S) plays a pivotal role in epidemiological analysis, phylodynamics, bionomics, and design of prophylactic measures to contain epidemics [38,29,45]. Epidemiological models are classified into Equation-based models (EBMs) (also called compartmental models) and Individual-based Models (IBMs) [38,29,45]. EBMs are classical approaches that use Ordinary Differential Equations (ODEs) to model transition of population

between disease states or compartments such as: Susceptible (S) → Exposed (E) → Infected (I) → Recovered (R) [6].

In contrast to EBMs, IBMs are descriptive rather than prescriptive models. Unlike EBMs that model the population as homogeneous aggregate compartments, IBMs model individuals in the population to explore heterogeneity, phylodynamics, antigenic drift, and other important epidemiological phenomena. Consequently, IBMs are most effective in epidemiological analysis of emergent diseases whose characteristics are not well understood [35,29,38]. Moreover, IBMs enable more vivid and intuitive temporo-geospatial visualization for various analyses.

The disadvantages of IBMs in the context of this research are: ① unlike human populations, the coarse resolution of surveillance data for waterfowl and limited data availability on behaviors of various species prevent generation of an IBM with realistic contact networks for waterfowl species [39]; ② the large number of parameters in IBMs poses challenges for validation; and ③ IBMs are computationally demanding [2,5,31,48] and can require 1000s of cores to simulate models with billions of individuals in short time frames [32,24,17,48].

3. Background on time warp

Time Warp is a popular optimistic synchronization protocol used for Parallel Discrete Event Simulation (PDES) [17]. A Time Warp synchronized PDES is organized as a set of asynchronous logical processes (LPs) or agents that represent the different physical processes being modeled. The LPs interact with each other by exchanging *virtual time* stamped events. Each LP processes its events by incrementing its local virtual time (LVT), changing its state, and generating new events. Although each LP processes local events in their correct timestamp order, events are not globally ordered. Causality violations are detected when an event with timestamp lower than the current LVT (a *straggler* event) is received. On receiving a straggler event a rollback mechanism is invoked to recover from the causality error. The rollback process recovers the LP's state prior to the causal violation, canceling the erroneous output events, and re-processing the events in their correct causal order. Each LP maintains a queue of state transitions along with lists of input and output events corresponding to each state to enable rollback recovery. A periodic garbage collection approach based on Global Virtual Time (GVT) is used to prune the queues by discarding history items that are no longer needed. The distributed simulation is deemed to have terminated when all the events in the system have been processed in their correct causal order.

4. SEARUMS: our prior research

Analyzing the epidemiological and socioeconomic impact of migratory waterfowl on global spread of avian influenza using computational approaches requires temporo-geospatial modeling and simulation of billions of waterfowl [35,32]. Equation-based models (EBMs) are computationally efficient but do not yield vital geospatial characteristics necessitating the use of Individual-based models (IBMs). However, as discussed in Section 2, the modeling issues and computational demands of IBMs prevent their use for analysis of billions of individuals, even with modern supercomputers [2,5,31,24]. Consequently, we utilize a novel, hybrid approach involving a combination of IBMs and EBMs for temporo-geospatial modeling [35,36,33,32].

4.1. Hybrid modeling approach

In our hybrid modeling approach, colocated birds of the same species are modeled as one aggregate entity called a *flock* [35,36,33,32]. Each flock represents only a single species because individuals modeled by it share the same epidemiological and ecological characteristics. Epidemiological properties of each flock were characterized using EBM approach while inter-flock epidemiological characteristics were modeled using the IBM approach. Aggregation of colocated individuals dramatically reduces the number of entities in the model thereby reducing computational demands without compromising necessary epidemiological and temporo-geospatial characteristics. Further discussion on our hybrid modeling approach is available in our prior publications [35,36,33,32].

4.2. Automatic model generation

The ecological models of migratory waterfowl are generated from aggregate satellite telemetry data on various waterfowl species and migratory corridors available as Geographic Information Systems (GIS) datasets. Fig. 7(a) shows an example of the GIS data for *Anas acuta* (Northern Pintail) from the Global Register of Migratory Species (GROMS) database [14]. Fig. 7(c) illustrates an example of spatially-explicit flocks (shown as circles) generated for agent-based simulation. Generic migratory corridors shown in Fig. 7(b) are suitably adapted to model migratory flyways for each waterfowl metapopulation as shown in Fig. 7(c). Migratory characteristics of various species obtained from Birdlife database [4] are used to determine temporal attributes for the flyways. The generated models are stored in an XML format for simulation-based epidemiological analysis. Details on methods used for model generation are discussed in our earlier publication [33].

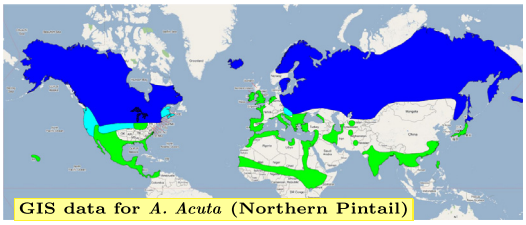
4.3. SEARUMS: Simulation & analysis environment

SEARUMSis a modeling, simulation, and analysis environment currently optimized to enable study and analysis of global epidemiology of avian influenza. It has been implemented in Java and enables graphical visualization of the model and simulation results as illustrated by Fig. 2. SEARUMSprovides an Agent-based Modeling (ABM) and a multithreaded Discrete Event Simulation (DES) infrastructure [35]. A `Waterfowl` agent models the ecological and epidemiological behaviors and characteristics of a *flock* (shown in Fig. 7(c)) using a discrete time Markov process shown in Fig. 8. Various epidemiological state transitions and inter-agent interactions are accomplished via virtual timestamped events as discussed in [35,32].

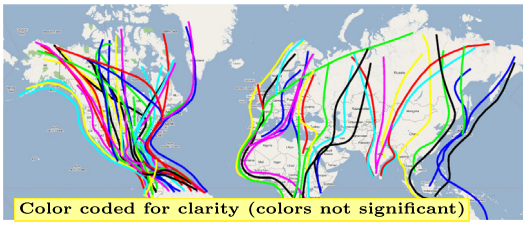
Spatially-explicit interactions between `Flock` agents are modeled using an `EcoArea` agent that represents the Earth's surface. The `EcoArea` agent receives updates from `Flock` agents upon changes to selected attributes such as: current coordinate, infection percentages, and population. The `EcoArea` components track agents and use the information to detect and trigger interactions between overlapping agents by scheduling virtual timestamped events as detailed in [35]. SEARUMSincludes capabilities for logging, charting, and tools to collate simulation results and conduct various epidemiological analyses.

4.4. Validation, analyses, & case studies

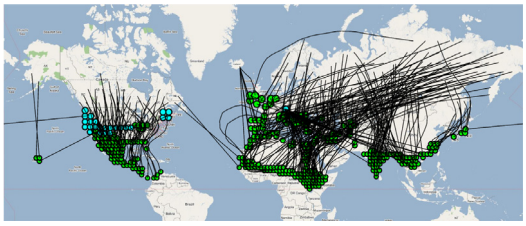
The modeling and simulation approach utilized in SEARUMShas been extensively validated using several case studies. The temporal characteristics of high risk outbreaks reported by the simulation were compared with outbreak timelines reported by World Health Organization (WHO). The temporo-geospatial data from our simulations closely correlated to major outbreaks reported by WHO as summarized in Fig. 9(a) and discussed in our publications [35,33,36]. The high risk outbreak regions reported by the simulation were consistent with those identified via surveillance [10]. As



(a) Raw GROMS data.



(b) Migratory corridors.



(c) Generated model.

Fig. 7. Surveillance GIS data sample, migratory corridors, and generated model with *flocks* of colocated individuals shown as circles. Detailed procedure for model generation is discussed in [33] and online at http://searums.org/esm13.

The SEIR ODEs used by the agent. Please refer to our earlier publications for further details [14, 10].

$$ds/dt = \lambda - [\mu + \lambda(t)]s(t)$$

$$d\lambda/dt = [v+\mu]\lambda(t)[R_0 s(t) - 1]$$

where s(t) is the proportion of population remaining susceptible at time t, and R_0 is the basic reproductive rate, μ is death rate, v is recovery rate, and λ(t) is infection rate.

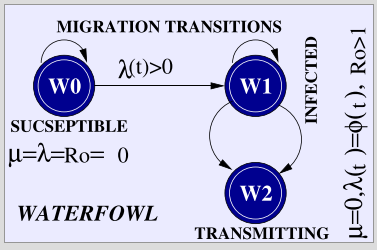
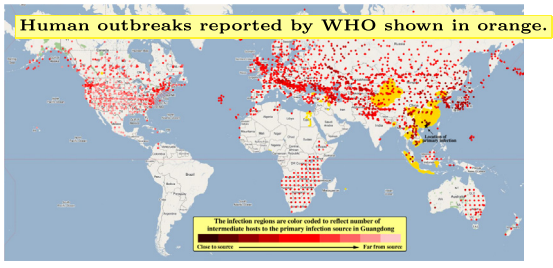


Fig. 8. *SEIR* Markov process for `Waterfowl` agent.

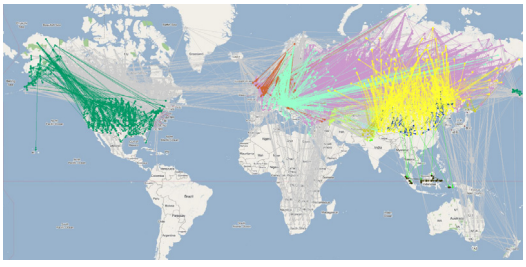
Outbreak Location	Outbreak Dates		Error (Days)
	Real–World	Simulated	
Indonesia	23–Jan–06	1–Jan06	22
Iraq/Iran	1–Mar–06	25–Mar–06	24
China	27–Apr–06	2–Apr–06	−25
Egypt	11–Oct–06	14–Sep–06	−27

Additional outbreak analysis data points are presented and discussed in our publications.

(a) Outbreak timeline (full list in [35]).



(b) Geospatial correlations [33].



(c) Phylogeographic annotation [31].

Fig. 9. Results from validation conducted using various case studies reported in our publications [35,36,33,32].

shown in Fig. 9(b), the high risk outbreak regions (in dark shades of red) showed strong correlation with actual human outbreaks as discussed in [33,32]. The infection pathways reported by the model were validated using a phylogeographic method using similar viral isolates (as defined by WHO) from various regions [32]. The validation discussed in [32] uses the fact that similar H5N1 isolates (with <1.5% difference) from waterfowl from two different countries indicate a direct infection pathway [32,47]. Readers are

referred to our earlier publications for epidemiological, antigenic, and prophylactic analysis along with sensitivity analysis and validation results [35,36,33,32].

5. SEARUMS++: The new parallel & distributed simulation environment

An architectural overview of the redesigned system called SEARUMS++ is shown in Fig. 10. It has been developed to enable optimistically synchronized PDES on distributed memory platform, in contrast to the earlier shared memory, conservatively synchronized infrastructure. SEARUMS++ has been developed by interfacing the Java-based GUI of SEARUMS with a Time Warp synchronized simulation infrastructure developed in C++, using a framework called WESE [30]. WESEwas chosen because its web-interface could be readily reused to interface with the existing Java modeling and visualization frontend. Furthermore the proposed investigations use traditional state saving method to recover from rollbacks due to the following two reasons. First, the state saving approach permitted the C++ implementation to closely mirror the initial implementation in Java, thereby facilitating effective verification and validation. Second, using reverse computation approach requires development of additional reverse processing logic (for rollback recovery) which introduces additional development effort. Given the complexities of our epidemic model, our research currently uses a state saving approach, with exploration of the alternative reverse computation based strategy as a future endeavor.

WESEprovides the infrastructure for developing a Time Warp synchronized parallel simulation and a C++ Application Program Interface (API) for developing agents. The actual agents that perform the core simulation-time activities are part of WESEfactories that are deployed on various compute nodes (via MPI) used for parallel simulation. A WESE`Factory` acts as an agent repository and as a Time Warp synchronized simulation kernel that uses MPI for exchanging events between factories. Fig. 10 illustrates the modules constituting a WESE`Factory`. The `communication subsystem` handles the tasks of interacting with remote SEARUMS++ clients and other WESEfactories via a custom application-layer protocol over TCP/IP connections. The `gateway` modules use the `communication subsystem` to provide the initial entry point to a WESE`Factory`. It creates a new `session manager` for each unique session initiated with the factory. The `session manager` performs the task of interacting with the `agent factory` to create the actual agents and locally establishes part of the Time Warp synchronized parallel simulation. The `simulation manager` then initiates a distributed simulation by interacting with `session managers` on the WESE`Factories` used in a simulation.

Agents in a single WESE`Factory` are part of the same process and are scheduled using a single thread of execution. Scheduling of events is performed using a shared Least-Time-Stamp-First (LTSF) event queue. Therefore, events exchanged between agents on the same WESE`Factory` never cause rollbacks. Conversely, inter-`Factory` events that are exchanged over the interconnect fabric (via MPI calls) give raise to straggler events resulting in rollbacks. WESEalso handles Global Virtual Time (GVT) based garbage collection and generation of optimistic I/O. Each `session manager` also performs the task of redirecting standard output from local simulations back to a SEARUMS++ client for visualization. A more detailed description of WESEand the process of developing a WESE`Factory` are available in the literature [30].

The graphical user interface and the WESE-based parallel simulation back-end have been coupled together using a Java-based interface layer. The `simulation manager` is the core module

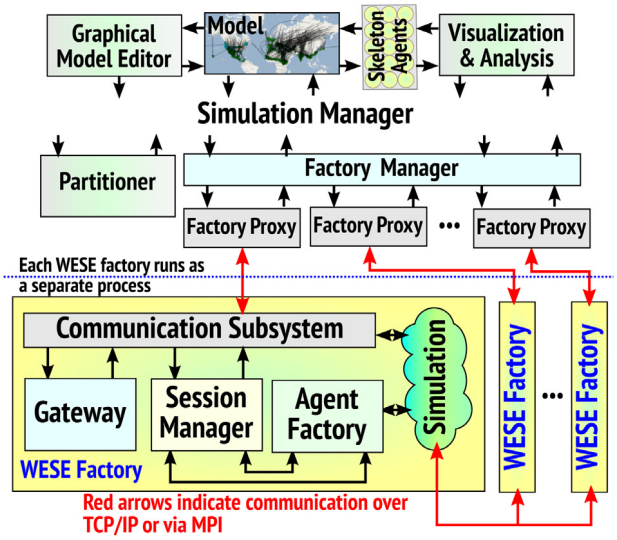


Fig. 10. Architectural overview of SEARUMS++.

that establishes, controls, and coordinates all the simulation related activities. It first interacts with the `partitioner` to suitably partition the model for parallel simulation. Then it uses the `factory proxy` modules to interact with various factories used for parallel simulation. Protocol details involved in communicating with a WESE`Factory` are handled by individual `factory proxy` modules. The `factory manager` coordinates the various `factory proxy` modules and eases interactions between them and the `simulation manager`. The `simulation manager` also performs the task of routing application-level messages to suitable visualization modules that update corresponding skeleton agents. The skeleton agents provide a dynamic (*i.e.,* during simulation), intuitive, graphical representation of the actual agents on various factories.

5.1. Agents in SEARUMS++

The `Waterfowl` agent models the ecological and epidemiological behaviors of a *flock* using characteristics generated from surveillance datasets as discussed in Section 4.2. The conceptual lifecycle model for a `Waterfowl` agent is shown in Fig. 8. As discussed in Section 4.1, each `Waterfowl` agent models a *flock* of colocated birds of the same species. This agent provides different configurations to model intra-flock and inter-flock infection spread [34]. Inter-flock infections occur when the agent migrates and comes in contact with any other agent in the model. The `Waterfowl` agent behaves as a classical Time Warp Logical Process (LP) introduced in Section 3. The temporal behaviors and life cycle processes of agents are coordinated by scheduling suitable timestamped events.

The spatial interactions between agents are modeled using `EcoArea` components that represent Earth's surface. `EcoArea` components receive update events from an agent initially and whenever it changes its attributes, such as: current coordinate, infection percentages, and population changes. The `EcoArea` components track agents in its purview by maintaining a list in its state. It uses the information to detect and trigger interactions between overlapping agents by scheduling timestamped events.

5.2. Spatial agents in SEARUMS++

SEARUMS++ provides two different `EcoArea` components that can be used in the two different configurations enumerated below. The two configurations were developed in an iterative

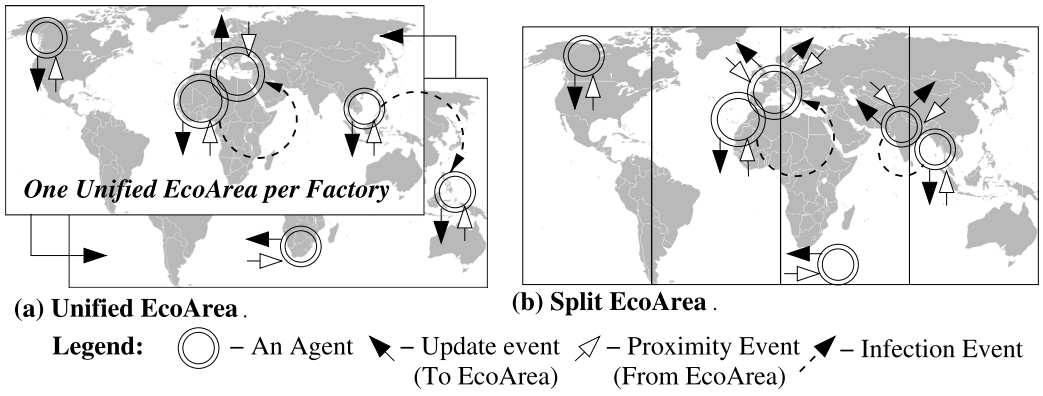


Fig. 11. Different simulation configurations involving `Unified` and `Split EcoArea` components. The figure shows some of the key events exchanged between agents and the `EcoArea` to model various life cycle activities as discussed in Section 5.2.

manner to address scalability and performance issues observed in each iteration. However, note that immaterial of the configuration used, the final application-level events are identical in all cases and consequently the simulation-results do not change. The two different `EcoArea` configurations explored in the design of SEARUMS++ were:

Configuration #1: Unified EcoArea: This component represents the complete surface of the earth as shown in Fig. 11(a). One unique component is instantiated on each compute node used for parallel simulation by the `simulation manager`. Agents partitioned to that node only send `update` events to their local `Unified EcoArea` to report changes in their location as agents migrate. Conversely, each `Unified EcoArea` component processes the update events and broadcasts aggregated updates to other `EcoAreas` to ensure coherence of agent information. These broadcasts may cause additional local `proximity` events to be generated by receiving `EcoAreas` to update local agents on its node. Agents receive `proximity` notifications only from their local `EcoArea`. Agents utilize the information in `proximity` events to directly schedule `infection` events to overlapping agents. The objective of this design was to minimize inter-`Factory` communication that ultimately plays a crucial role in causing rollbacks. However, as discussed in Section 8, this configuration did not yield good scalability. Therefore, we propose the alternative approach discussed below to circumvent the bottlenecks observed in this configuration.

Configuration #2: Split EcoArea: This component represents a specific rectangular region of the Earth's surface. Areas represented by `Split EcoArea` components are distinct and do not overlap. One unique `Split EcoArea` component is created per WESE `Factory`. Furthermore, the `partitioner` (see Fig. 10) distributes agents based on their geographical coordinates to the corresponding `Factory`. When `Split EcoArea` is used in a simulation, agents communicate only with the `EcoAreas` they overlap. Fig. 11(b) illustrates this configuration involving four factories. Note that as agents move they appropriately register and unregister themselves from the corresponding `Split EcoArea`.

5.3. Motivation for proxy agents

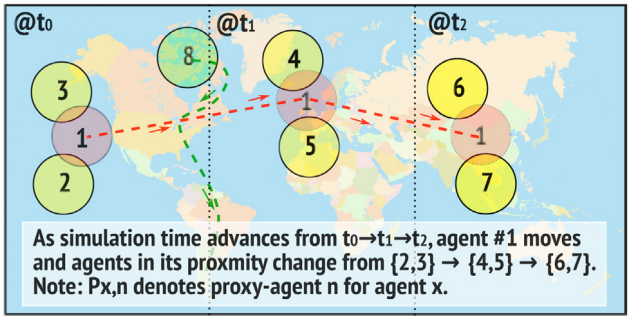
The Time Warp synchronized PDES infrastructure of SEARUMS++ provided performance improvement over the initial conservatively synchronized, Java-based multithreaded kernel. However, the PDES did not efficiently scale as the number of mobile agents in the model was increased. The root cause of the issue was identified to be large number of Inter-Process Messages (IPMs) that increased synchronization overheads and degraded performance. IPMs arise because agents are initially partitioned to different compute nodes (on the compute cluster used for PDES) and they do not *physically* move (the C++ object for an agent is fixed on a compute node), but only move *logically* by changing their latitude and longitude values stored in their state. However, as illustrated in Fig. 12(a) and (b), when agents logically move they have to interact with agents on a different compute node giving rise to IPMs. Consequently, as simulation time advances from t_0 to t_2 ($t_0 < t_1 < t_2$), the communication patterns change increasing IPMs.

IPMs negatively impact synchronization resulting in degraded scalability and performance as illustrated by the charts in Fig. 13. The graphs in Fig. 13 also illustrate the strong correlation (Pearson correlation coefficient $r = 0.98$, p-value <0.0001) between IPMs (Fig. 13(a)) and rollbacks (in Fig. 13(b)) as well as a strong correlation ($r = 0.96$, p-value <0.0001) between inter-process messages and simulation execution time (Fig. 13(c)).

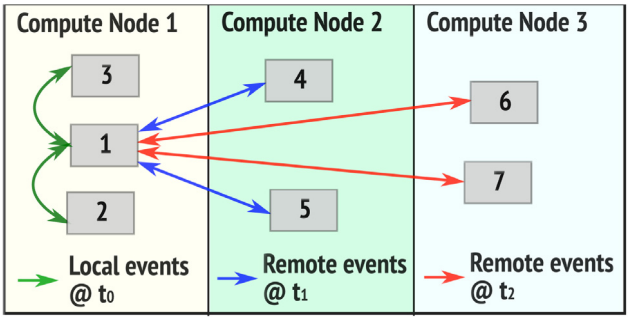
5.4. Single Active Proxy (SAP) method

In continuation with the discussions in Section 5.3, the volume of Inter-Process Messages (IPMs) arising due to movements of agents had to be reduced to improve scalability and performance. Reducing IPMs requires that interacting agents must be predominantly on the same compute node—implying that as agents *logically* migrate they must be correspondingly relocated to a different compute node. Accordingly, the notion of *proxy* agents was introduced in the simulation. Proxy agents are agents that are automatically created for each mobile agent on each compute node used for PDES as shown in Fig. 12(c). As an agent logically moves across `EcoArea` boundaries, it deactivates itself after activating an appropriate proxy on a different compute node. Therefore, only one proxy-agent is active at any given time and consequently this strategy is called Single Active Proxy (SAP) approach.

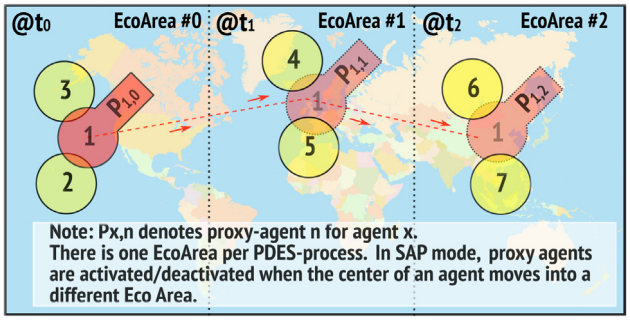
A simplified pseudo code for the SAP approach is shown in Algorithm 1. Initially, one deactivated proxy agent is automatically created on each PDES-process for every mobile agent in a given model. Deactivated agents do not perform any operations and remain dormant in the simulation. However, based on the initial geographical location of an agent, the appropriate proxy activates itself and performs the normal operations of the agent. When the agent migrates across its local `EcoArea`, the active proxy first deactivates itself (sets a flag in its state) *logically* removing itself from the simulation. The proxy then schedules an activation event (which includes the current state) to the appropriate proxy object on a different `EcoArea`. For example when agent #1 in Fig. 12(c) moves from `EcoArea #0` to `EcoArea #1` proxy agent $P_{1,0}$ deactivates itself while activating proxy agent $P_{1,1}$. The activated proxy then resumes life cycle activities for the agent. Subsequent state changes are not communicated back to the deactivated proxy.



(a) Conceptual view of changes in communication patterns due to movements of an agent.

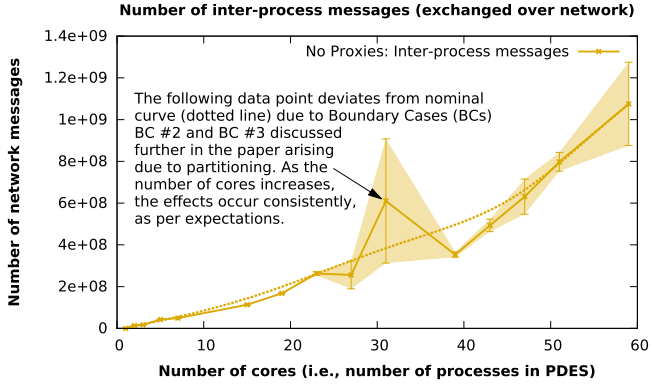


(b) Corresponding change from local to remote events as C++ objects for agents are on the same compute node.

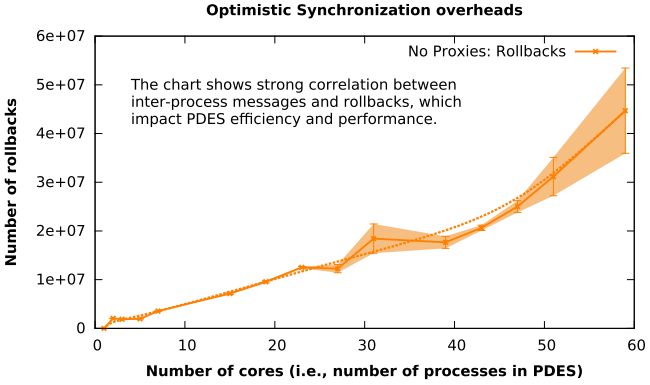


(c) Overview of Single Active Proxy (SAP) approach with 3 `EcoAreas` on 3 processes.

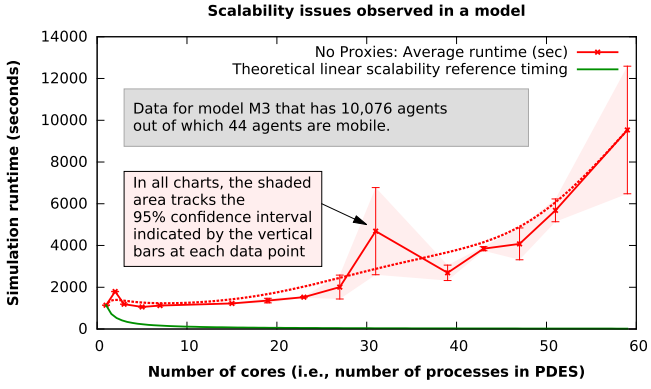
Fig. 12. Overview of problem (increase in Inter-Process Messages (IPMs) due to conceptual movement of agents) and the Single Active Proxy (SAP) approach to address the issue.



(a) Inter-Process Messages (IPMs).



(b) Rollbacks.



(c) PDES execution time.

Fig. 13. Experimental data (*without proxies* from model `M3` discussed in Section 8) illustrating lack of scalability even with `SplitEcoArea` configuration.

Algorithm 1: Single Active Proxy (SAP) Approach

```
1  begin initialization(state)
2      if inLocalEcoArea( state→latitude, state→longitude) then
3          state→isActive = true;
4          registerWithEcoArea();
5          scheduleLifeCycleEvents();
6      else
7          state→isActive = false;
8      end if
9  end initialization
10 begin processEvent(state, event)
11     if state→isActive then
12         proessEvent(event);
13         if ! inLocalEcoArea( state→latitude, state→longitude) then
14             state→activeProxy = unregisterFromEcoArea();
15             activateProxy(state, state→activeProxy);
16             state→isActive = false;
17         end if
18     else
19         if isProxyEvent( event) then
20             activate();
21             state→isActive = true;
22         else
23             rescheduleEvent(state→activeProxy, event→recvTime + DELTA);
24         end if
25     end if
26 end processEvent
```

The dormant proxies have an extra role of forwarding events (as events may have already been scheduled for the dormant proxy) to the active proxy. Activation, deactivation, and forwarding of events are performed using sub-simulation cycles denoted by DELTA in Algorithm 1. Specifically, simulation time is defined as a double precision value and fractional time (DELTA = 0.0001) is added to meet the causal constraint that all events sent by an agent must have a time stamp greater than the agent's LVT. In other words, in order to forward an event e with timestamp $t_{recv}(e)$, a deactivated proxy creates and schedules copy of the event with timestamp $t_{recv}(e)$ + DELTA to the active proxy. The tools for post-simulation analysis of simulation logs ignore the fractional DELTA values in log messages. Consequently, the filtered simulation logs used for various analysis are identical in all cases.

5.5. Boundary cases in SAP

The Single Active Proxy (SAP) approach provided significant improvement in scalability and performance over the base case as discussed in Section 8. However, the simulations continued to experience some performance degradation as the number of compute nodes used for simulation was increased. Experimental analysis indicated that the SAP approach still experienced increased inter-process messages and rollbacks when an agent spans two or more partitions, as enumerated in the following three different Boundary Cases (BCs):

BC #1: The first case occurs when agents transitioning from one `EcoArea` to another across PDES-process boundaries as illustrated for agent #1 in Fig. 14. In this transitional period only one proxy is active in SAP and has to interact with agents in two different PDES-processes (namely agents #3 and #4) resulting in increased Inter-Process Messages (IPMs) and degraded performance.

BC #2: The second case occurs when a mobile agent is resting but happens to spans across two `EcoAreas` as illustrated by the set of agents {#6, #7, #8} in Fig. 14. Since agents span two or more `EcoAreas`, it gives rise to copious amounts of Inter-Process Messages (IPMs) which results in degraded performance. The frequency of this scenario steadily increases as the number of `EcoAreas` or partitions increases. Increase in partitions results in thin or small `EcoAreas` thereby increasing the probability that an agent spans two `EcoAreas`. This issue conspicuously manifests itself even without SAP (as expected) as highlighted by the chart in Fig. 13(a).

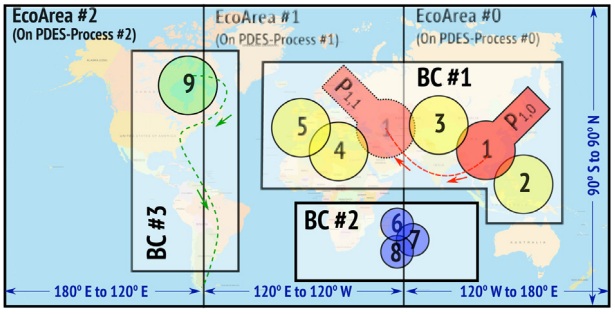


Fig. 14. Illustration of three Boundary Cases (BCs) that impact SAP approach.

BC #3: The third BC occurs when an agent's movement oscillates between two or more partitions, similar to the pathway of agent #9 shown in Fig. 14. In this scenario, proxy agents often experience cascading rollbacks which degrades performance.

6. Multiple Active Proxy (MAP)

The Multiple Active Proxy (MAP) approach is proposed to extend SAP approach and address the three boundary cases (BCs) discussed in Section 5.5 to provide a more comprehensive solution. The MAP approach extends the SAP approach by permitting multiple proxies to be active at the boundary cases. The pseudo code in Algorithm 2 provides additional details about the MAP implementation. In MAP approach, each active proxy handles interactions with other agents on its local partition, thereby minimizing Inter-Process Messages (IPMs) that are exchanged in SAP mode. Moreover, each proxy agent also performs various life cycle tasks by scheduling events to itself. Replication of life cycle tasks at each active proxies does increase the net number of events in the simulation. However, these events are local events that do not significantly degrade performance.

Algorithm 2: Multiple Active Proxy (MAP) Approach

```
1  begin initialization(state)
2      ecoAreas=ecoAreas(state→latitude, state→longitude);
3      if localEcoArea ∈ ecoAreas then
4          state→isActive = true;
5          registerWithEcoArea();
6          scheduleLifeCycleEvents();
7      else
8          state→isActive = false;
9      end if
10 end initialization
11 begin processEvent(state, event)
12     if state→isActive then
13         prevEcoAreas=ecoAreas(state→latitude, state→longitude);
14         proessEvent(event);
15         if ! inLocalEcoArea( state→latitude, state→longitude) then
16             state→activeProxy = unregisterFromEcoArea();
17             state→isActive = false;
18         else
19             newEcoAreas=ecoAreas(state→latitude, state→longitude) -
               prevEcoAreas;
20             if newEcoAreas != φ then
21                 activeProxies(newEcoAreas);
22             end if
23         end if
           // Synchronize active proxy states
24         activeProxyStateCoherence();
25     else
26         if isProxyEvent( event) then
27             activate();
28             state→isactive = true;
29         else
30             rescheduleEvent(state→activeProxy, event→recvTime + DELTA);
31         end if
32     end if
33 end processEvent
```

At the end of handling local interactions, the active proxies exchange messages (which are IPMs) to consistently update the states of all active proxies using a straightforward protocol. Each proxy maintains a summary of interactions which includes list of neighbors infected and amount of infection received from neighbors. The summary information is sent via timestamped events to other active proxies, if any. Upon receiving summary information from another active proxy, each proxy appropriately updates its local state. At the end of the phase of event exchanges all proxies have the same, consistent state. The straightforward protocol involves $O(n_a^2)$ events, where n_a is number of active proxies per agent. However, n_a is typically a small value (in the range 2–5) and therefore is not a significant overhead.

Once an agent no longer spans multiple partitions, relevant proxies deactivate themselves and the simulation continues to proceed in SAP mode. MAP minimizes the number of Inter-Process Messages (IPMs) at boundary cases thereby improving upon the efficiency gained by utilizing SAP. In this context, it must be noted that the final resulting state in both MAP and SAP approaches is the same as that of the original agent's state. Consequently, the results from the simulations used for various analyses are identical in all cases.

7. Related research

The proposed research is a novel integration of several major topics in Modeling and Simulation (M&S), including: agent-based epidemic modeling, spatially-explicit models, and optimistic PDES. Several investigations have been reported on these topics and this section compares the proposed research with some of the closely related investigations. Readers are referred to the references and literature for a more comprehensive survey of related works [35,33].

Agent-based models (ABMs) have been employed for epidemiological analyses for various diseases [8,22]. Parker and Epstein [22] discuss the issues involved in design and development of a Java-based, Global-Scale Agent Model (GSAM) distributed platform for epidemiological modeling using over 6 billion interacting agents. Similar to GSAM, the proposed research avoids "physical" process migration but unlike GSAM the proposed research involves "logical" process migration using proxy agents. However, in contrast to agents in GSAM, the mobile agents in our model explicitly embody migration and inter-agent interactions without relying on contact matrices or contact networks [22]. Moreover, the agents in our research are different in that they do not represent a single entity but a collection of collocated entities, striking a better balance between model resolution and execution costs. Our modeling approach is useful for M&S of 21 billion migrating waterfowl using surveillance and satellite telemetry data.

The use of ABMs with explicit agent mobility distinguishes the proposed research from those reported by Barrett et al. [2], Bisset et al. [5], and Perumalla et al. [24]. These three investigations use a reaction–diffusion approach involving contact graphs for epidemic modeling. The use of contact graphs or social networks for epidemic modeling has been discussed by other researchers as well [9,19,15]. Recently, Yeom et al. [48] discuss the use of their simulator called `EpiSimdemics` to simulate epidemic progression using contact networks for 280.4 million individuals on 352 K core-modules on a Cray XE6/XK7. Their implementation uses two-level hierarchy to aggregate several individuals into `PersonManagers` and several areas where they interact into `LocationManagers` to balance various overheads. The contact networks-based model proposed by Yeom et al. have locations and schedules that permit static analysis and static partitioning to obtain effective load balance across the 352 K core-modules. The proposed research uses a very different model than Yeom et al. [48]—although the migratory flyways for waterfowl are specified in our models, the contacts between agents are not preordained but discovered during simulation based on probabilistic movements of agents. Yeom et al. use synthetic contact networks to model and simulate interactions between agents. Consequently, the partitioning and parallelization approaches for our model need to be different than the static approaches proposed by Yeom et al. However, similar to the aforementioned investigations we aim to utilize parallel simulations to accelerate performance of epidemic simulations.

The proposed Single Active Proxy (SAP) and Multiple Active Proxies (MAP) approach discussed in Sections 5.4 and 6 can be viewed as "logical" process migration in contrast to the "physical" process migration that has been extensively investigated in conservative as well as optimistic PDES [17,23]. There are a few aspects that distinguish traditional process migration from our approach. First, all necessary proxies are created on different PDES-processes at the beginning of simulation. Consequently, there are more agents in the simulation than there are in the model—*i.e.,* if a model has m mobile, i immobile agents and the simulation is run on p processors ($p \geq 1$), then the total number of agents in the simulation is $m + i + m(p - 1)$, where $m(p - 1)$ are proxy agents. In contrast, in traditional process migration there would be only $m + i$ agents in the simulation. Second, the proxy agents remained fixed on the PDES-processes in which they were created and they logically activate or deactivate themselves (via a flag in their state) to mimic migration. Consequently, as far as the underlying simulation kernel is concerned, there is absolutely no changes occurring to the simulation, which is a stark contrast to physical migration in which new agents are typically created and the unused ones are destroyed.

Unlike SAP that essentially accomplishes "logical" process migration, strategies involving physical migration of logical processes have been proposed for dynamic load balancing [25]. However, unlike the application-specific, logical migration accomplished by SAP, almost all of the efforts reported in the literature focus on providing a generic infrastructure for physical process migration. Consequently, the earlier investigations involve extensions to the underlying simulation kernel or using a new runtime infrastructure. On the other hand, the proposed investigations focus on model-level extensions that can be applied to both conservative and optimistic PDES. Furthermore, SAP and MAP are agnostic to the underlying simulation kernel.

`Charm++` is a widely used language and runtime library for distributed computing that performs migration of "chares", that are synonymous to an agent in our research, with the primary objective of accomplishing dynamic load balancing [1,3]. Several simulators such as ROSS [26] and `EpiSimdemics` [48] have been developed using `Charm++`. `Charm++` performs physical migration of chares while SAP does not physically migrate proxy agents. SAP logically activates and deactivates proxies on different nodes (when agents migrate) with the objective to reduce inter-process communication and consequently synchronization overheads, rather than accomplishing load balancing as with `Charm++`. Furthermore, MAP permits multiple proxies (akin to having multiple copies of one "chare") to be active while in `Charm++` only a single copy of a "chare" is active. Load balancing methods have also been used in the context of shared memory multithreaded parallel simulators. Khaligh et al. discuss the use of a dynamic load balancing to improve performance of multithreaded simulations of adaptive transaction level models (TLMs) [18]. The multithreading approach is similar to the method used in the initial version of our simulator. However, the revised version, namely SEARUMS++, does not use multithreading and is designed for distributed memory systems.

The SAP and MAP approaches are distantly related to "ghosting" in multi-resolution simulations in which multiple representations

of entities are maintained [43]. The term ghosting is used to refer to scenarios similar to MAP in which multiple distinct entities perform operations that are conceptually associated with just one entity. However, MAP approach is different than the concept of "ghosting" in multi-resolution simulations in that the agents are not at different resolutions and in fact strive to be identical copies of each other. Furthermore, unlike "ghosts" that typically participate throughout a simulation, in MAP approach proxies dynamically activate and deactivate themselves. Similar to MAP approach, migration of portions of shared states for adaptive load management of multi-agent systems has also been explored [21]. Similarly, distribution and maintenance of consistent states and sub-states is an integral aspect of the Data Distribution Management (DDM) service in the High Level Architecture (HLA) [46]. In contrast, the MAP approach does not subdivide states of agents but aims to replicate complete, consistent state resulting in multiple, identical proxy agents in the simulation. Furthermore, interactions between agents use an on-demand request–response approach rather than the publish–subscribe approach used in HLA. Maintaining, identical coherent states for proxy agents also distinguishes MAP approach from the *passive* agent sharing approach used in RepastHPC [7]. In RepastHPC copies of agents are passive in that changes to the copies are not transmitted to the actual agent [7]. Conversely, in MAP changes to a proxy are used to reach an eventually consistent state for the subset of currently *active* proxies (and not all proxies) associated with a given agent.

Optimistic PDES on distributed memory platforms using reverse computation approaches to recover from rollbacks have been reported [24,26]. Unlike the reverse computation based approaches, the proposed investigations use traditional state saving method to recover from rollbacks for the following two reasons. First, the state saving approach permitted the C++ implementation to closely mirror the initial implementation in Java, thereby facilitating effective verification and validation. Second, using reverse computation approach (such as the one used in ROSS [26]) requires development of suitable reverse processing logic (for being used during rollback recovery) which requires additional analysis and effort. Given the complexities of our epidemic model, our research currently uses a state saving approach, with exploration of the alternative reverse computation based strategy as a future endeavor.

8. Experiments

The experiments conducted to evaluate the effectiveness of the proposed methods to enable efficient parallel simulation were performed using a range of different models. Table 1 shows the number of mobile and immobile agents constituting four models generated using the methods introduced in Section 4.2. The models `M1`, `M2`, and `M3` have a varying number of agents at different scales of detail used for sensitivity analysis. Unlike the first three models in Table 1, model `M4`, the primary model used for various analyses presented in Section 4.4, serves as a stress-test case in which all agents are mobile.

All the experiments were conducted on a distributed memory supercomputing cluster running Red Hat Enterprise Linux (RHEL 6.5). Each compute node had two hex-core Intel® Xeon™ X5650 CPUs @ 2.67 GHz (with 12 MB L2 cache) yielding 5333 bogomips on each of the 12 cores. Each compute node had 48 GB of RAM averaging to 4 GB per core. The nodes were interconnected using QDR infiniband interconnect. Note that the experiments were performed using the non-GUI mode supported by SEARUMS to avoid GUI overheads. Furthermore, the Java frontend was executed on a separate compute node and model-logs were turned off to eliminate any resource contentions or I/O overheads for all performance tests reported in this section. The experiments did not use any Time Warp optimizations, such as lazy cancellation, to eliminate their influence on experimental results and to clearly distinguishing benefits of the SAP and MAP approaches that focus on reducing Inter-Process Messages (IPMs) generated by the model. Furthermore, SAP and MAP approaches operate at the model level permitting them to be used with other optimistic or conservatively synchronized kernels. Minimizing Time Warp related optimizations permits the experimental results to provide an estimate of their applicability to other PDES infrastructures.

Table 1
Summary of models used in experiments.

Model ID	Number of agents		Total
	Mobile	Stationary	
M1	44	2628	2672
M2	44	6411	6455
M3	44	10032	10076
M4	3088	0	3088

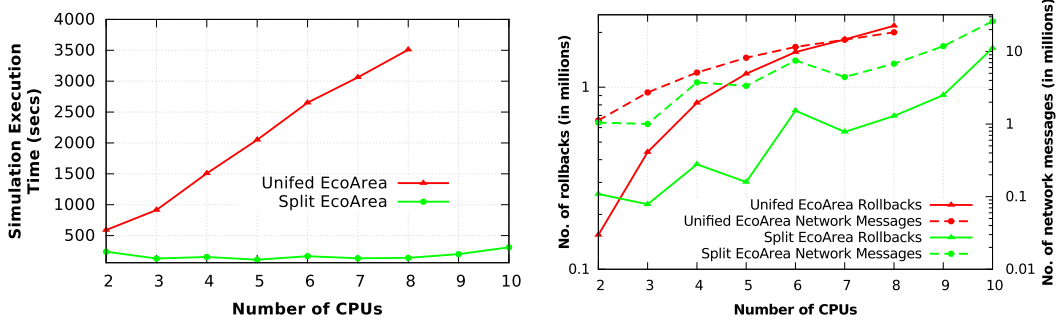
8.1. Unified versus split `EcoArea` comparisons

The initial redesign utilized `Unified EcoArea` agents (introduced in Section 5.2) reflecting the characteristics of the original shared-memory implementation discussed in Section 4.3. However, as illustrated by the charts in Fig. 15, the `Unified EcoArea` configuration did not efficiently scale and perform. Experiments involving larger number of cores were abandoned since run times started to steadily increase. The root cause of the bottleneck was the significant increase in broadcast messages between `EcoAreas` in this configuration as evident from the corresponding curve in Fig. 15(b). As the number of nodes used for parallel simulation is increased, the number of events exchanged between `Unified EcoAreas` grows, increasing the number of Inter-Process Messages (IPMs). Increase in IPMs increases probability of straggler messages (see Section 5.3 for correlation between IPMs and rollbacks) thereby increasing number of rollbacks (*i.e.,* synchronization overheads) as seen in Fig. 15(b) and thus degrading the performance.

On the other hand, the `Split EcoArea` method do not involve broadcasting of events to update pertinent state information. Instead, as described in Section 5.2, the agents directly schedule events to appropriate `Split EcoAreas` depending on their current geographic location. This feature significantly restricts the necessary cross `EcoArea` interactions thereby improving performance as evidenced by the charts in Fig. 15. The `Split EcoArea` experiments used a vertical partitioning scheme in which the surface of the earth was divided into vertical strips as shown in Fig. 11. This strategy provided us the best load balance due to nature of the migratory patterns that are predominantly vertical, *i.e.,* in North ↔South direction. Prior to the vertical partitioning, we pursued dividing the surface into a 2-dimensional grid. However, the grid partitioning scheme resulted in diminished scalability and performance and consequently was abandoned. Furthermore, as discussed in Section 5.3 and illustrated by the slightly concave curve in Fig. 15(a), the `Split EcoArea` configuration without proxies (*i.e.,* did not use SAP or MAP approaches) did not scale well as the computational resources were increased.

8.2. Experimental evaluation of SAP & MAP

The experiments involving Single Active Proxy (SAP) and Multiple Active Proxy (MAP) were conducted to assess weak and strong scalability of the two approaches. The weak scaling experiments were conducted a synthetic model with 10 mobile agents migrating horizontally from the western to eastern



(a) Execution time.

(b) Rollbacks & messages.

Fig. 15. Data for `Unified` and `Split` `EcoArea` for `M1`.

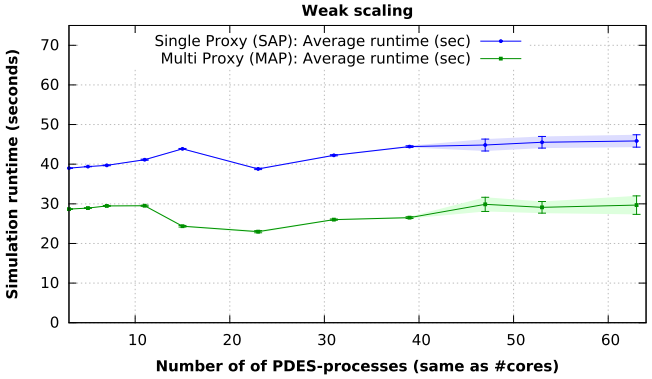


Fig. 16. Simulation runtimes for SAP and MAP approaches to analyze weak scaling characteristics.

hemisphere. The `Split EcoArea` configuration with vertical partitioning scheme was used to trigger the use of proxies as the agents migrate. In addition, 200 immobile agents were created on each PDES-process. All the flocks were seeded with an initial infection to trigger various epidemiological activities to maintain about the same workload on each PDES-process. Fig. 16 illustrates the simulation runtime for the SAP and MAP approaches using a varying number of cores. The error bar at each of the data points indicates the 95% confidence interval computed from the 10 runs. The lightly shaded region tracks the confidence interval to highlight statistical significance of the observations. The average simulation times for the SAP and MAP approaches range from 39–45 s and 22–29 s respectively. Variations in timings arise due to mobile agents migrating across the vertical partitions in the different configurations. The simulation time slightly increases with increase in number of cores due to additional overheads of GVT-based garbage collection. The charts in Fig. 16 illustrate that the SAP and MAP configurations yield typical weak scaling characteristics.

The strong scaling experiments for evaluating the proposed Single Active Proxy (SAP) and Multiple Active Proxy (MAP) approaches were conducted using the `Split EcoArea` configuration. The observations collated from experiments conducted using a range of configurations for different models (see Table 1) are shown by the charts in Figs. 17 and 18. The *x*-axis of all the charts corresponds to the number of parallel PDES-processes (*i.e.,* MPI processes), with each process running on an independent core. The solid lines on each one of the charts track average values obtained from 10 runs. The corresponding dotted lines show the curve fitted for the observations to emphasize deviations from general trends due to the occurrence of Boundary Cases (BCs) as discussed in Section 5.5. The error bar at each of the data points indicates the 95% confidence interval computed from the 10 runs and the lightly shaded region tracks the confidence interval to highlight statistical significance of the observations.

In all the charts in Figs. 17 and 18, the `base case` configuration corresponds to simulations with `Split EcoArea` conducted without the use of proxies. However, the base case curves are not included in the charts for model `M4` (the "stress test" case) because the configuration without proxies was practically unusable due to long simulation execution times (over 24 h with over 32 cores) and the experiments were abandoned. The charts for model `M1` in Fig. 17 already highlight the effectiveness of the Single Active Proxy (SAP) and Multiple Active Proxies (MAP) approaches. However, for the small `M1` model both the SAP and MAP approaches are pretty close in runtime for most configurations, with the MAP approach performing about 10% to 15% better. The small difference is expected because the MAP approach operates as SAP except for boundary cases.

The graphs for model `M2` in Fig. 17 and model `M3` in Fig. 18 illustrate the advantages of SAP and MAP approach. Furthermore, the advantages of MAP are a bit more prominent in these two models. The SAP approach consistently outperforms the base case by about 2× while the MAP approach provides another 15% to 25% performance boost on top of SAP. In these two models the overall effect of MAP is still muted because the models do not have many mobile agents in them. Nevertheless, the MAP approach provides much better scalability that SAP for both `M2` and `M3` models. On the other hand, the SAP approach starts experiencing degradation in scalability around 20 cores and 40 cores for models `M2` and `M3` respectively. The degradation in scalability occurs sooner for `M2` than for `M3` because `M2` is a smaller model with less workload and with increased availability of compute power the model optimistically advances only to suffer from increased rollbacks, as evidenced by the charts in Fig. 17.

The advantages of MAP approach are more pronounced in the case of model `M4` that has a large number of mobile agents. The abundance of mobile agents increases the chances of Boundary Cases (BCs) as the number of PDES-processes is increased. Unlike the SAP method, the MAP approach continues to provide much better scalability and about 20% to 25% performance improvement in the configurations with up to 48 cores. Reprising the discussion from earlier paragraph, the base case data for model `M4` are not shown in Fig. 18 as the experiments were abandoned due to extremely long simulation times that exceeded 24 h in several configurations.

The experimental results shown in Figs. 17 and 18 also illustrate the strong correlation between increase in Inter-Process Messages (IPMs), the number of rollbacks, and simulation execution time. Note that the count of IPMs includes anti-messages used by the Time Warp protocol for optimistic synchronization. For example, with model `M4` in SAP mode, on an average each rollback resulted in 18.21 anti-messages out of which 7.46% were non-local, *i.e.,* dispatched to other PDES-processes. On the other hand, with

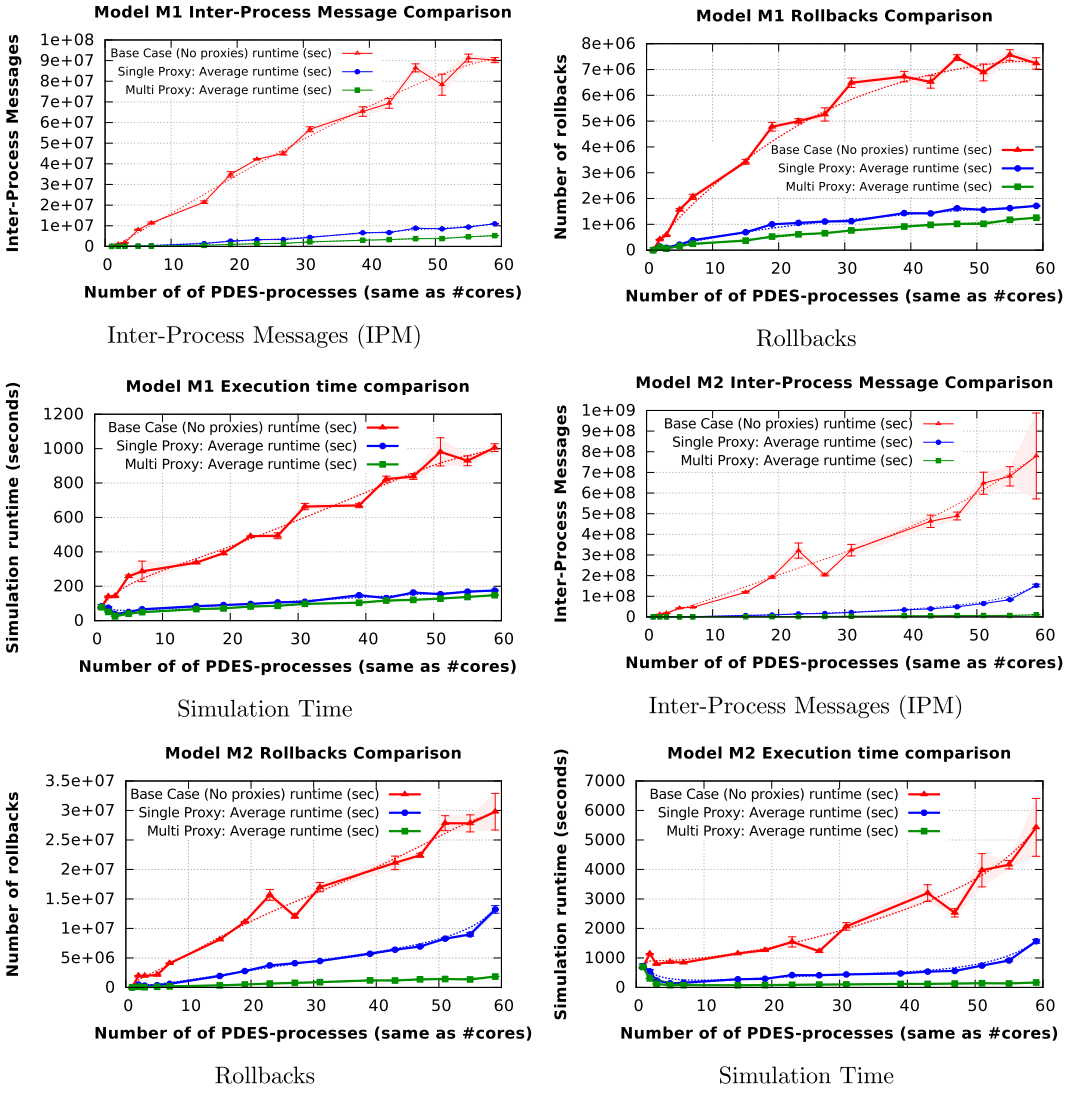


Fig. 17. Comparison of Inter-Process Messages (IPM), rollbacks, and simulation execution time for the models M1 and M2 shown in Table 1 in the following three configurations: no proxies (base case), Single Active Proxy (SAP) approach, and Multiple Active Proxies (MAP) approach. The corresponding dotted lines show the curve fitted for the observations to emphasize deviations from general trends due to the occurrence of Boundary Cases (BCs) as discussed in Section 5.5. Note that number of PDES-processes is the same as the number of CPU-cores used for simulation (one PDES-process per CPU-core).

model M4 in MAP mode, on an average each rollback resulted in 30.88 anti-messages of which 2.11% were non-local. In conjunction with chart on number of rollbacks from Fig. 18, the data shows that the MAP approach experiences fewer but deeper rollbacks with very few non-local anti-messages when compared to the SAP approach. The number of rollbacks is fewer because the MAP approach is effective in eliminating a large fraction of IPMs by using more than one active proxy. Furthermore, the MAP approach improves upon SAP by further reducing IPMs and consequently the fraction of non-local anti-messages is lower.

8.2.1. Analysis of plateauing of strong scalability

The experimental results in Figs. 17 and 18 show that improvements in runtime are observed only up to about 48 cores and then slowly increase after about 50 cores even for the larger models M3 and M4. The reasons for this behavior are:

1. Consistent with Amdhal's Law [13], with a fixed size model, as the number of cores is increased the speedup does not increase beyond a certain threshold due to the upper bound on inherent parallelism available in the model.
2. As the number of cores is increased from 2 to 64, the number of agents per node for model M4 (that shows improvements up to 48 cores) reduces on an average from 1544 (*i.e.*, 3088÷2) agents per PDES-process to just ∼48 (*i.e.*, 3088 ÷ 64) agents per PDES-process. This causes the PDES-processes to over optimistically progress into the future only to get rolled-back as straggler events arrive over the network. Consequently, the net speedup realized is limited as the computation performed per node decreases while communication overheads start to dominate.
3. More importantly, the geography of the model combined with vertical partitioning strongly influences and limits the effectiveness of increasing number of partitions. As summarized in Fig. 19, due to the curvature of Earth the area between longitudes rapidly decreases as latitude increases. During summer several species migrate north to latitudes ∼75°N (*i.e.*, northern Russia, Alaska, Canada, etc.), where the distance between latitudes is just ∼17 miles. With 12 partitions the width of a vertical partition is 510 miles (360 ÷ 12 × 17 miles) and about 255 miles (360 ÷ 24 × 17 miles) with 24 partitions. With flock sizes spanning 50–100 miles based on available GROMS data (see Section 4.3 for model generation details), most flocks

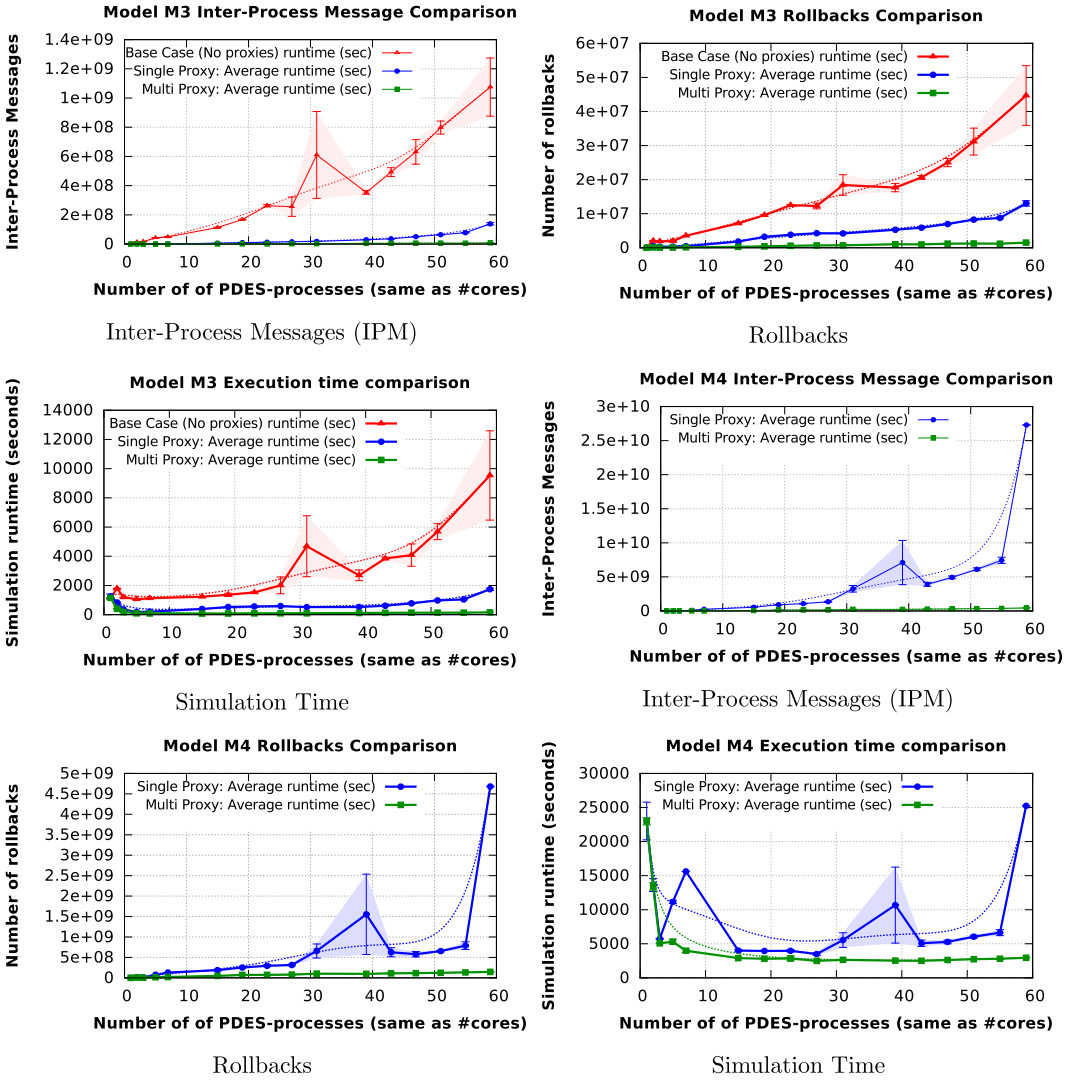
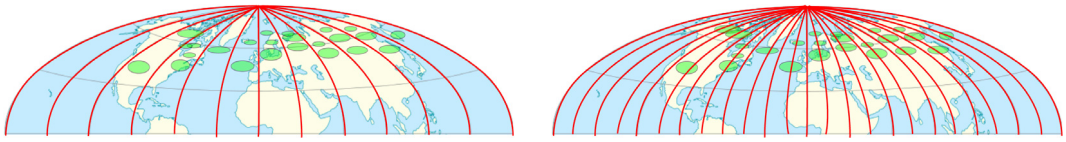


Fig. 18. Comparison of Inter-Process Messages (IPM), rollbacks, and simulation execution time for the models M3 and M4 shown in Table 1 in the following three configurations: no proxies (base case), Single Active Proxy (SAP) approach, and Multiple Active Proxies (MAP) approach. The corresponding dotted lines show the curve fitted for the observations to emphasize deviations from general trends due to the occurrence of Boundary Cases (BCs) as discussed in Section 5.5. Note that number of PDES-processes is the same as the number of CPU-cores used for simulation (one PDES-process per CPU-core).

fit within the partitions allocated for each PDES-process and consequently fewer Inter-Process Messages (IPMs) are needed to model various interactions. However, with 60 partitions, the width of a vertical partition decreases to approximately 100 miles (360 ÷ 60 × 17 miles). Consequently, the proportion of flocks spanning two partitions dramatically increases requires a much larger number of IPMs. In other words, as the number of partitions is increased the Earth's surface area partitioned to them decreases (see Fig. 19(b)) and more flocks span across partitions, resulting in BC #2 scenario, requiring increased Inter-Process Messages (IPMs) to model various interactions. The increase in IPMs is evident, particularly in the SAP approach, where the IPMs sharply increase after about 50 cores (see Fig. 18). Increase in IPMs exacerbates synchronization overheads (see Section 5.3 for correlation between IPMs and synchronization overheads) and degrades speedup and limits its scalability. Unlike the SAP approach, the MAP approach minimizes number IPMs used in boundary cases. However, MAP does not completely eliminate IPMs arising from BC #2 because multiple proxies still have to exchange events to synchronize their state (see Section 6). Therefore, with increase in partitions beyond a threshold, the number of IPMs grows and negatively impacts speedup. Consequently, as illustrated by charts in Figs. 17 and 18, the inherent geospatial nature of the model hinders strong scalability once the number of cores is increased beyond a certain threshold.

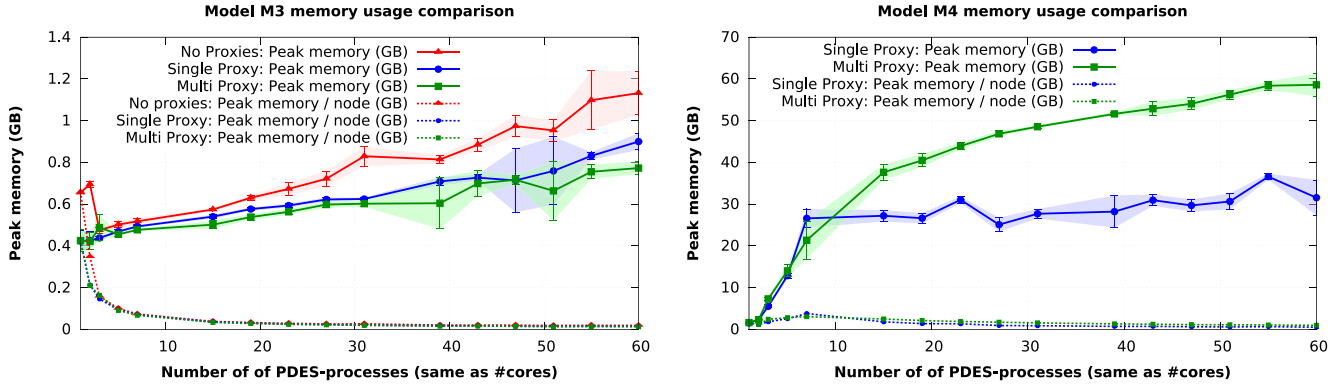
8.2.2. Analysis of memory overheads of proxies

The charts in Fig. 20 illustrate a comparison of the peak memory usage by Single Active Proxy (SAP) and Multiple Active Proxies (MAP) approaches. The solid lines in the chart show the peak memory footprint as reported by the job monitoring system on the supercomputing cluster. The peak memory is the sum of peak memory used by all the PDES-processes. The dotted lines in the charts show the average peak memory footprint on a per-node basis. The per-node memory footprint was computed by dividing total peak memory by the number of PDES-processes used for simulation. The data in Fig. 20(a) shows that the memory usage for model M3 is comparable between the three approaches. The base case that does not use proxies has a slightly higher footprint because increased synchronization overheads (see increased rollbacks in Fig. 17) cause additional events and states to be maintained in order to recover from causal violations. The memory footprint for the models with proxies is not pronounced because the number of mobile agents in M3 is much fewer when compared



(a) 12 partitions. (b) 24 partitions.

Fig. 19. Illustration of degradation in speedup due to decreasing area between longitudes and increasing number of cores which causes flocks to span multiple partitions requiring more Inter-Process Messages (IPMs). Increase in IPMs results in increased synchronization overheads (see correlation discussion in Section 5.3) and degradation of speedup. In order to highlight the scenario, the illustration uses a Hammer projection (that reduces distortion towards poles) instead of the popular Mercator projection (where area near poles is exaggerated in size) used for other figures.



(a) Model M3. (b) Model M4.

Fig. 20. Comparison of peak memory footprint between no proxy (base case), Single Active Proxy (SAP), and Multiple Active Proxies (MAP) approaches. The dotted lines show peak memory usage on a per-node basis, computed by dividing total peak memory by the number of PDES-processes.

to immobile agents. Recollect that proxies are used only for mobile agents and therefore with few mobile agents, the number of proxies is also few. Moreover, the advantages of proxying outweigh their memory footprint—*i.e.,* about 1.2 kB per state copy and memory for inter-proxy events in the MAP approach.

In contrast, for model M4 where all of the agents are mobile, the additional memory usage of the MAP approach is evident, particularly as the number of PDES-processes is increased. With 60 PDES-processes, the memory foot print for SAP and MAP approach is 31.5 (∼0.525 GB/node) and 58.6 (∼0.98 GB/node) respectively, yielding about 1.86× difference. The difference in memory footprint corresponds to the peak number of additional proxies that are active in the MAP approach. Note that the actual number of proxies that are active per agent varies (depending on number of PDES-processes used as discussed in Section 8.2.1) throughout the simulation. Nevertheless, as illustrated by the runtime charts in Fig. 18, the MAP approach provides 15%–57% improvement over SAP, trading-off memory to reduce runtime.

9. Conclusions

The application of simulation-based analysis using spatially explicit, agent-based models is gaining momentum for epidemiological analysis of emergent diseases such as avian influenza [38,29,45]. We have developed a modeling, parallel simulation, and analysis environment called SEARUMS++ to meet the computational demands of agent-based models. This paper described in the issues involved in the design and development of a Time Warp synchronized Parallel Discrete Event Simulation (PDES) infrastructure of SEARUMS++. The different design alternatives that were incrementally developed and empirically evaluated to identify the optimal simulation configuration were discussed.

The spatially-explicit nature of the models posed several challenges in achieving scalable and efficient PDES, necessitating new approaches for modeling spatial interactions using `Split EcoArea` agents as discussed in Section 5.2. The `Split EcoArea` approach experienced scalability issues as the number of mobile agents in the model was increased. Consequently, the investigation also focused on identifying and addressing the scalability and ensuing performance degradation. Exploratory investigations (presented in Section 5.3) identified that the large volume of Inter-Process Messages (IPMs) exchanged over the network was magnifying synchronization issues resulting in degraded scalability.

The investigations proposed and assessed the effectiveness of using *proxy* agents on each process for each agent. Initially, only one proxy was permitted to be active at any given time resulting in a Single Active Proxy (SAP) approach. The SAP approach provided over 200% performance improvement as indicated by the experimental results discussed in Section 8. However, the SAP approach did not address bottlenecks arising due to three different Boundary Cases (BCs) as discussed in Section 5.5. Consequently, a Multiple Active Proxy (MAP) approach was proposed and assessed.

The Multiple Active Proxy (MAP) approach extends SAP approach and permits multiple proxy agents to be active only in boundary case scenarios while operating in SAP mode otherwise. The MAP mode permits multiple active proxies to handle local interactions occurring on the same process with fewer Inter-Process Messages (IPMS) to maintain consistent states. Experimental evaluation of the MAP approach discussed in Section 8 indicates that the MAP approach provides more reliable scalability along with a performance boost of 15% to 57% on top of SAP. The SAP and MAP approaches are implemented at the model-level using the default infrastructure provided by the PDES kernel. Since the proposed solutions do not depend on any special support from the underlying PDES kernel, they can be readily implemented in any infrastructure, including conservatively synchronized PDES.

The experiments indicate that optimistic synchronization methodology effectively utilizes the latent parallelism in the model to yield improved scalability and performance. The optimal simulation configuration consistently yielded about 200% improvement and was scalable for a wide range of model sizes highlighting its effectiveness. The optimized PDES significantly

reduces simulation execution times, easing exploratory analyses, and highlights the current and future importance of parallel simulations in epidemiology, bionomics, and related fields.

Acknowledgment

This work was funded in part by the Ohio Supercomputer Center (PMIU0110-2).

Appendix A. Supplementary data

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.jpdc.2016.04.004.

References

[1] B. Acun, A. Gupta, N. Jain, A. Langer, H. Menon, E. Mikida, X. Ni, M. Robson, Y. Sun, E. Totoni, L. Wesolowski, L. Kale, Parallel programming with migratable objects: Charm++ in practice, in: SC14: International Conference for High Performance Computing, Networking, Storage and Analysis, 2014, pp. 647–658. http://dx.doi.org/10.1109/SC.2014.58.

[2] C.L. Barrett, K.R. Bisset, S.G. Eubank, X. Feng, M.V. Marathe, Episimdemics: An efficient algorithm for simulating the spread of infectious disease over large realistic social networks, in: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC'08, IEEE Press, Piscataway, NJ, USA, 2008, pp. 37:1–37:12.

[3] M.A. Bhandarkar, L.V. Kalé, E.d. Sturler, J. Hoeflinger, Adaptive load balancing for MPI programs, in: Proceedings of the International Conference on Computational Science-Part II, ICCS'01, Springer-Verlag, London, UK, UK, 2001, pp. 108–117.

[4] BirdLife International, BirdLife Data Zone, October 2012.

[5] K.R. Bisset, J. Chen, X. Feng, V.A. Kumar, M.V. Marathe, Epifast: A fast algorithm for large scale realistic epidemic simulations on distributed memory systems, in: Proceedings of the 23rd International Conference on Supercomputing, ICS'09, ACM, New York, NY, USA, 2009, pp. 430–439. http://dx.doi.org/10.1145/1542275.1542336.

[6] F. Brauer, C. Castillo-Chavez, Mathematical Models for Communicable Diseases, SIAM, 3600 Market Street, Philadelphia, PA 19104-2688, USA, 2013.

[7] N. Collier, M. North, Parallel agent-based simulation with repast for high performance computing, Simulation 89 (10) (2013) 1215–1235. http://dx.doi.org/10.1177/0037549712462620.

[8] J.M. Epstein, Modelling to contain pandemics, Nature 460 (2009) 687. http://dx.doi.org/10.1038/460687a.

[9] N.M. Ferguson, D.A.T. Cummings, C. Fraser1, J.C. Cajka, P.C. Cooley, D.S. Burke, Strategies for mitigating an influenza pandemic, Nature 442 (2006) 448–452. http://dx.doi.org/10.1038/nature04795.

[10] T.L. Fuller, M. Gilbert, V. Martin, J. Cappelle, P. Hosseini, K.Y. Njabo, S.A. Aziz, X. Xiao, P. Daszak, T.B. Smith, Predicting hotspots for influenza virus reassortment, Emerg. Infect. Dis. 19 (4) (2013). http://dx.doi.org/10.3201/eid1904.120903.

[11] M. Gilbert, D.U. Pfeiffer, Risk factor modelling of the spatio-temporal patterns of highly pathogenic avian influenza (HPAIV) H5N1: A review, Spat. Spat.-Temporal Epidemiol. 3 (3) (2012) 173–183.

[12] M. Gilbert, X. Xiao, J. Domenech, J. Lubroth, V. Martin, J. Slingenbergh, Anatidae migration in the western palearctic and spread of highly pathogenic avian influenza H5N1 virus, Emerg. Infect. Dis. 12 (11) (2006). http://dx.doi.org/10.3201/eid1211.060223.

[13] A. Grama, G. Karypis, V. Kumar, A. Gupta, An Introduction to Parallel Computing, second ed., Addison-Wesley, 2003.

[14] GROMS, Global Register of Migratory Species (GROMS): Summarising Knowledge about Migratory Species for Conservation, July 2013. URL: http://groms.gbif.org/.

[15] M.E. Halloran, N.M. Ferguson, S. Eubank, J. Ira, M. Longini, D.A.T. Cummings, B. Lewis, S. Xu, C. Fraser, A. Vullikanti, T.C. Germann, D. Wagener, R. Beckman, K. Kadau, C. Barrett, C.A. Macken, D.S. Burke, P. Cooley, Modeling targeted layered containment of an influenza pandemic in the United States, Proc. Natl. Acad. Sci. USA 105 (12) (2008) 4639–4644. http://dx.doi.org/10.1073/pnas.0706849105.

[16] M. Imai, T. Watanabe, M. Hatta, S.C. Das, M. Ozawa, K. Shinya, G. Zhong, A. Hanson, H. Katsura, S. Watanabe, C. Li, E. Kawakami, S.Y.M. Kiso, Y. Suzuki, E.A. Maher, G.N.Y. Kawaoka, Experimental adaptation of an influenza H5 HA confers respiratory droplet transmission to a reassortant H5 HA/H1N1 virus in ferrets, Nature 486 (2012) 420–428. http://dx.doi.org/10.1038/nature10831.

[17] S. Jafer, Q. Liu, G. Wainer, Synchronization methods in parallel and distributed discrete-event simulation, Simul. Model. Pract. Theory 30 (1) (2013) 54–73.

[18] R.S. Khaligh, M. Radetzki, A dynamic load balancing method for parallel simulation of accuracy adaptive TLMs, in: Specification Design Languages, FDL 2010, 2010 Forum on, 2010, pp. 1–6. http://dx.doi.org/10.1049/ic.2010.0141.

[19] I.M. Longini, A. Nizam, S. Xu, K. Ungchusak, W. Hanshaoworakul, D.A.T. Cummunings, M.E. Halloran, Containing pandemic influenza at the source, Science 309 (5737) (2005) 1083–1087. http://dx.doi.org/10.1126/science.1115717.

[20] D. Normile, Avian influenza: Evidence points to migratory birds in H5N1 spread, Science 311 (5765) (2006) 1225. http://dx.doi.org/10.1126/science.311.5765.1225.

[21] T. Oguara, D. Chen, G. Theodoropoulos, B. Logan, M. Lees, An adaptive load management mechanism for distributed simulation of multi-agent systems, in: Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications, 2005. DS-RT 2005 Proceedings. 2005, pp. 179–186. http://dx.doi.org/10.1109/DISTRA.2005.9.

[22] J. Parker, J.M. Epstein, A distributed platform for global-scale agent-based models of disease transmission, ACM Trans. Model. Comput. Simul. 22 (1) (2011) 2:1–2:25. http://dx.doi.org/10.1145/2043635.2043637.

[23] S. Peluso, D. Didona, F. Quaglia, Supports for transparent object-migration in PDEs systems, J. Simul. 6 (2012) 279–293.

[24] K.S. Perumalla, S.K. Seal, Discrete event modeling and massively parallel execution of epidemic outbreak phenomena, Simulation 88 (7) (2012) 768–783. http://dx.doi.org/10.1177/0037549711413001.

[25] P. Peschlow, T. Honecker, P. Martini, A flexible dynamic partitioning algorithm for optimistic distributed simulation, in: 21st International Workshop on Principles of Advanced and Distributed Simulation, 2007. PADS'07, 2007, pp. 219–228. http://dx.doi.org/10.1109/PADS.2007.6.

[26] J. Peter, D. Barnes, C.D. Carothers, L.V. Kalé, Extreme scale optimistic parallel discrete event simulation with dynamic load balancing, in: Proceedings of the 2014 Winter Simulation Conference, 2014, pp. 4125–4126.

[27] President's Council of Advisors on Science and Technology, Report to the president on engineering the influenza vaccine production enterprise to meet the challenges of pandemic influenza, August 2010.

[28] D. Prosser, L. Hungerford, R.M. Erwin, M.A. Ottinger, J.Y. Takekawa, E. Ellis, Mapping avian influenza transmission risk at the interface of domestic poultry and wild birds, Front. Public Health 1 (28) (2013). http://dx.doi.org/10.3389/fpubh.2013.00028.

[29] L.L. Pullum, O. Ozmen, Early results from metamorphic testing of epidemiological models, in: 2012 ASE/IEEE International Conference on BioMedical Computing, BioMedCom, 2012, pp. 62–67. http://dx.doi.org/10.1109/BioMedCom.2012.17.

[30] D.M. Rao, Study of dynamic component substitution (Ph.D. thesis), University of Cincinnati, 2003.

[31] D.M. Rao, Accelerating parallel agent-based epidemiological simulations, in: Proceedings of the 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, SIGSIM PADS'14, ACM, New York, NY, USA, 2014, pp. 127–138. http://dx.doi.org/10.1145/2601381.2601387.

[32] D.M. Rao, Enhancing epidemiological analysis of intercontinental dispersion of h5n1 viral strains by migratory waterfowl using phylogeography, BMC Proc. 8 (6) (2014) 1–10. http://dx.doi.org/10.1186/1753-6561-8-S6-S1.

[33] D.M. Rao, A. Chernyakhovsky, Automatic generation of global agent-based model of migratory waterfowl for epidemiological analysis, in: Proceedings of the 27th European Simulation and Modelling Conference (ESM'2013), EuroSis, Lancaster University, Lancaster, UK, 2013, p. e1. best paper award.

[34] D.M. Rao, A. Chernyakhovsky, V. Rao, SEARUMS: Studying epidemiology of avian influenza rapidly using simulation, in: Proceedings of ICMHA'07, Berkeley, CA, USA, 2007, pp. 667–673.

[35] D.M. Rao, A. Chernyakhovsky, V. Rao, Modeling and analysis of global epidemiology of avian influenza, Environ. Modell. Softw. 24 (1) (2009) 124–134. http://dx.doi.org/10.1016/j.envsoft.2008.06.011.

[36] D.M. Rao, A. Chernyakhovsky, V. Rao, Human-in-the-Loop Simulations, Springer, 2011, pp. 153–174. (Chapter Analyzing Global Epidemiology of Diseases Using Human-in-the-Loop Bio-Simulations).

[37] R. Rappuoli, P.R. Dormitzer, Influenza: Options to improve pandemic preparation, Science 336 (6088) (2012) 1531–1533. http://dx.doi.org/10.1126/science.1221466.

[38] B. Roche, J. Drake, P. Rohani, An agent-based model to study the epidemiological and evolutionary dynamics of influenza viruses, BMC Bioinformatics 12 (1) (2011) 87. http://dx.doi.org/10.1186/1471-2105-12-87.

[39] A. Saltelli, P. Annoni, How to avoid a perfunctory sensitivity analysis, Environ. Modell. Softw. 25 (12) (2010) 1508–1517.

[40] S. Sonnberg, R.J. Webby, R.G. Webster, Natural history of highly pathogenic avian influenza H5N1, Virus Res. 178 (1) (2013) 63–77.

[41] S. Swarup, S.G. Eubank, M.V. Marathe, Computational epidemiology as a challenge domain for multiagent systems, in: Proceedings of The Thirteenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS), AAMSAS'14, ACM, New York, NY, USA, 2014, pp. 1173–1177.

[42] K. Tharakaraman, R. Raman, K. Viswanathan, N.W. Stebbins, A. Jayaraman, A. Krishnan, V. Sasisekharan, R. Sasisekharan, Structural determinants for naturally evolving H5N1 hemagglutinin to switch its receptor specificity, Cell 1016 (10) (2013). http://dx.doi.org/10.1016/j.cell.2013.05.035.

[43] A. Tolk, Engineering Principles of Combat Modeling and Distributed Simulation, Wiley, 2012.

[44] USDA, Assessment of introduction pathway for novel avian influenza virus into north america by wild birds from eurasia, Tech. rep., United States Department of Agriculture (USDA)'s Animal & Plant Health Inspection Service (APHIS) Department of Veterinary Services and Centers for Epidemiology and Animal Health, available online (May 2013).

[45] E.M. Volz, K. Koelle, T. Bedford, Viral phylodynamics, PLoS Comput. Biol. 9 (3) (2013) e1002947. http://dx.doi.org/10.1371/journal.pcbi.1002947.

[46] J. Wang, T. Zheng, A hybrid multicast-unicast assignment approach for data distribution management in HLA, Simul. Model. Pract. Theory 40 (0) (2014) 39–63.

[47] H.M. Wilson, M.R. Petersen, M.G. Sexson, Relation of highly pathogenic avian influenza (h5n1) prevalence to migration patterns of pacific common eiders nesting in northwest Alaska: Summary report, 2007-northwest Alaska, Tech. rep., US Fish and Wildlife Service and US Geological Survey, available online (September 2007).

[48] J.-S. Yeom, A. Bhatele, K. Bisset, E. Bohm, A. Gupta, L.V. Kale, M. Marathe, D.S. Nikolopoulos, M. Schulz, L. Wesolowski, Overcoming the scalability challenges of epidemic simulations on blue waters, in: Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium, IPDPS'14, IEEE Computer Society, Washington, DC, USA, 2014, pp. 755–764. http://dx.doi.org/10.1109/IPDPS.2014.83.

Dhananjai M. Rao is currently an assistant professor in the Computer Science and Software Engineering (CSE) department at Miami University, Oxford, OHIO, USA. He graduated from University of Cincinnati with M.S. and Ph.D. in 2000 and 2003 respectively. He has over 8 years of industry experience working on various web-technologies and cloud computing. His research interests include modeling, simulation, high performance computing, and bioinformatics.