Many Resident Task Computing in Support of Dynamic Ensemble Computations

Jonathan Ozik,*† Nicholson T. Collier,*† Justin M. Wozniak†‡
*Global Security Sciences Division, Argonne National Laboratory, Argonne, IL, USA
†Computation Institute, University of Chicago and Argonne National Laboratory, Chicago, IL, USA
‡Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA

***Abstract*—Modern agent-based models (ABMs) that can simulate large populations are increasingly used to answer important questions in a variety of topic areas. For these models to serve as useful electronic laboratories, they require a battery of experimental analyses, including calibration and validation, sensitivity analysis, optimization, data assimilation, and multi-model integration. These analyses are typically run as workflows that coordinate ensembles of simulations on open science high-performance computing (HPC) systems, enabling large numbers of concurrent simulations. The execution model of these workflows is driven by the algorithmic logic, which can involve arbitrary loops and recursive behavior in pursuit of model convergence. In this paper, we describe a *many resident task computing* framework that coordinates ABM ensembles on open science HPC systems, driven by a pluggable, stateful optimization engine. While this model challenges conventional notions about workflows consisting of many run-to-completion tasks, we show how it enables rapid prototyping of many parameter search and optimization strategies. Our focus here is on ABMs and optimization, but these techniques are more widely applicable to any black-box scientific code and adaptive parameter space characterization. Ultimately, the goal is to democratize the use of HPC resources by enabling non-expert researchers to take advantage of the extreme scale systems that will be available in the next few years.**

I. Introduction

High-performance agent-based models (ABMs) are a promising method to simulate a variety of complex systems, including the spread of infectious diseases and community-based healthcare interventions [1], [2], critical materials supply chains [3], and land-use and resource management [4], [5]. Realistic ABMs can include large numbers of parameters (>100) that govern the structural (e.g., social networks), behavioral, and other dynamical elements of the systems being modeled. In a real system, these parameters are not immediately known but may be derived from an *inverse modeling* process. In this approach, approximate parameters are applied to a simulated model and model outputs are compared with empirical real-world outcomes. Then, the parameters are iteratively refined according to one of many possible algorithms.

The highly non-linear relationship between ABM input parameters and model outputs prevent the use of many mathematical optimization techniques. These relatively efficient methods cannot be applied due to the complex interactions among the parameters, as well as feedback loops and emergent behaviors. Consequently, a more flexible approach is required in which workflows of large numbers ("ensembles") of simulations need to be run. Also needed are sophisticated statistical algorithms, which adaptively refine model parameters through the analysis of recently generated simulation results and launch new simulations, are needed. These simulation ensemble workflows do not fit the MapReduce model or simple directed acyclic graph (DAG) specifications.

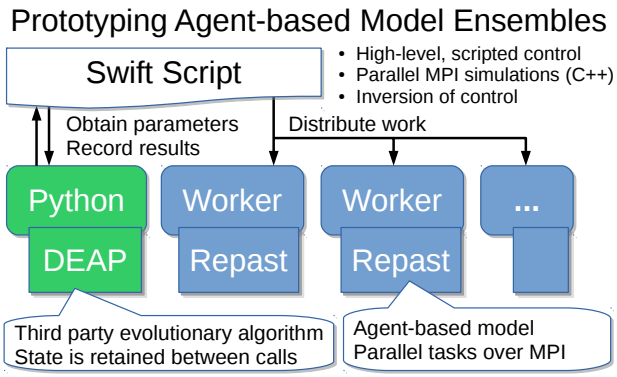

Figure 1: Overview of ABM ensemble framework in Swift/T.

In this work, we present a general pattern for solving these problems using the Swift/T dataflow language [6]. We demonstrate the use of the third-party Distributed Evolutionary Algorithms in Python (DEAP) toolkit [7] to implement an evolutionary algorithm (EA) that controls the dynamic workflow logic. We show how this was implemented with the use of *resident* Python tasks in Swift/T and simple queue-based interfaces for passing parameters and simulation results.

Our framework, shown in Figure 1, enables the user to plug in parameter exploration algorithms and scientific applications (e.g., ABMs). Thus, researchers in various fields who may not be parallel programming experts can simply select from a variety of previously developed search algorithms and run computational experiments on their scientific application without explicitly performing parallel programming. A key feature of this approach is that neither the search algorithm nor the scientific application is modified to fit our framework. Rather, the overall parameter search task is initiated by Swift and does not exit until search completion. This is implemented in a reusable way by connecting the parameter generation and output registration methods to interprocess communication (IPC) mechanisms that allow these values to be exchanged with Swift/T.

This mechanism is a novel form of inversion of control (IoC). Swift/T instantiates the search algorithm, which then provides simulation parameters back to Swift/T (over IPC, without returning). These parameters are distributed to worker processes for execution. Upon completion, the model outputs are registered back to the search algorithm, which provides more parameters until a convergence criterion is satisfied.

The mechanism also relies on new "many *resident* task computing" (MRTC) capabilities that extend the notion of many-task computing (MTC). This allows running tasks to effectively suspend, waiting for queries. Mixing resident tasks with traditional run-to-completion tasks is a powerful programming model that supports the development of calibrated and validated realistic ABMs that can be used as *electronic laboratories* to answer important research and policy questions.

The remainder of this paper is organized as follows. In §II we describe the ABM and adaptive parameter search application area, including related methods. In §III we describe our programming model and its implementation. In §IV we report performance numbers for the complete application workflow. In §V we summarize our contributions and outline future work.

II. Agent-based models

Agent-based modeling and simulation (ABMS) is a method of computing the potential system-level consequences of the behaviors of sets of individuals [8]. ABMS allows modelers to specify the individual behavioral rules for each agent; to describe the circumstances in which the individuals reside; and then to execute the rules in order to determine possible system-level results. Agents themselves are individually identifiable components that usually represent decision makers at some level. Agents often are capable of some level of learning or adaptation ranging from simple parameter adjustment to the use of neural networks, evolutionary algorithms, and market models.

As more complicated models of larger complex systems are developed, HPC resources are increasingly used to run the variety of computational experiments needed to develop validated models that support decision-making. ABMS studies typically require the execution of many model runs to account for stochastic variation in model outputs as well as to explore the possible range of parameter dependent outcomes, making them well suited to running on HPC resources. For example, ABMs of infectious diseases have included large numbers of individuals and households to create detailed activity-based models of the propagation of different diseases, including influenza [1], and community associated MRSA [2]. Various scientific workflows are required to calibrate and analyze these large-scale models: adaptive parametric studies; large-scale sensitivity analyses and scaling studies; optimization and metaheuristics; inverse modeling; uncertainty quantification; and data assimilation.

A. ABM ensembles

While statistical adaptive parameter search techniques have been a generally fruitful approach for combining ensemble mathematical (e.g., compartmental) models and empirical observations, for example in infectious disease modeling [9], [10], [11], we also see that the recent Ebola outbreak has exposed some limits to the predictive power of this combination of ensemble modeling [12]. The possible reasons for this are many, but some of the simplifying assumptions inherent in the compartmental models that are used for these infectious disease studies might be at issue. Compartmental models use differential equations relating aggregate variables (e.g., the fractions of the population that are susceptible, infected, or recovered/removed) to derive the dynamics of disease progression in a population. But such models are not able to capture "complex social networks and the direct contacts between individuals, who adapt their behaviors." [13]. By developing more realistic models in the form of ABMs, the complexity, for example of the inter-agent and biological-social interactions inherent in many infectious diseases, can be encapsulated in the specification of processes such as agent activities and decision-making, agent interactions over social networks, demographic and geographic heterogeneity, and agent adaptation and learning.

As such, statistical adaptive parameter search techniques that had been applied to simpler modeling paradigms are increasingly being used with ABMs [14]. With the large number of simulation runs required and the use of realistic ABMs, these analyses will need to be run as workflows that coordinate ensembles of simulations on open science HPC systems, where the execution logic of the workflows will be defined by the specifics of the chosen adaptive parameter search algorithm.

B. Related adaptive parameter search techniques

Depending on the aims of a computational experiment, different adaptive parameter search techniques are appropriate.[1] For stochastic optimization of an objective function, techniques such as simulated annealing [18] and adaptive mesh techniques [19] can be used. Alternatively, when the goal of model evaluation is to explore and identify important regions of parameter space, as opposed to finding only a global optimum, evolutionary algorithms, such as genetic algorithms [20], can be used to iteratively "grow" such a collection of parameters. Similarly, approximate Bayesian computation [21], [22] techniques can be used to generate distributions of parameter values consistent with observations.

For systems where periodic data are expected, Bayesian data assimilation approaches can be used. These include methods such as ensemble Kalman filtering [23] and particle filters [24], [25]. Data assimilation allows for the periodic combination of ensembles of model outputs and observations to give the best current estimate, along with estimates of uncertainties, for a

[1]We note that there exist static parameter search techniques (e.g., full factorial design [15], Latin hypercube sampling [16], Morris method [17]) that *a priori* determine the sampling from a parameter space. While these can be useful for some purposes, they are not adaptive and do not require complex workflow logic and hence are not the focus of this paper.

system under study. This enables back-casting and forecasting of model results.

Many of these techniques are being actively developed and are implemented as free and open source libraries in popular programming languages. As indicated earlier, rather than requiring the reimplementation of these algorithms in Swift/T, the goal of our framework is to be able to have these libraries directly control large-scale HPC workflows, thereby making them more accessible to a wider range of researchers and, at the same time, enable them to run at HPC scales.

III. Programming Model

Our programming model is designed to satisfy the requirements described in §II. We desire a high-level programming model that allows us to coordinate calls to ABMs as well as various control and analysis scripts over a scalable, MPI-based computing infrastructure. For this more basic aspect, we use Swift/T (although other workflow systems could suffice). The key aspect of this work is the need for additional features to support the adaptive parameter search techniques. These features include location-aware scheduling, resident tasks, and non-trivial IPC. We then present an application of our framework that uses DEAP to drive an EA over ABMs built with the Java-based Repast Simphony [26] toolkit.

A. Location-aware many-task scheduling

MTC workloads, on the one hand, generally allow the *scheduler* a great deal of leeway in determining where tasks will execute. Bag-of-tasks workloads, for example, are the most lenient, allowing tasks to execute anywhere in any order. Programming models such as MPI, on the other hand, give the *programmer* total control over execution locality.

Swift/T strikes a balance between these two extremes with its *location annotation*. By default, tasks can execute on any worker process, but the programmer has the option of specifying the annotation with `@location=L f()`, where `f()` is the task and `L` is a location value. A location value is constructed from an MPI rank r with optional accuracy and strictness qualifiers. (Swift/T features allow a hostname to be translated to one or more MPI ranks.) The accuracy may be RANK, specifying the process with rank r, or NODE, specifying any process that shares the same network host with r. The strictness may be SOFT, allowing the task to run anywhere in the system if there is nothing else to do at a given point in time, or HARD, specifying that the scheduler should wait until the location constraint can be satisfied (even at the expense of maintaining idle processors).

The location features in Swift/T were originally added for data-intensive workloads [27]. These provide a novel model for best effort, data-aware scheduling, when data is stored on the compute nodes. Compute node-resident storage systems that advertise data locations can be exploited by these programming features. In this work, we extend the utility of this feature by using it to target program state instead of bulk data. By keeping program state resident, we avoid any cost associated with approaches that depend on data serialization.

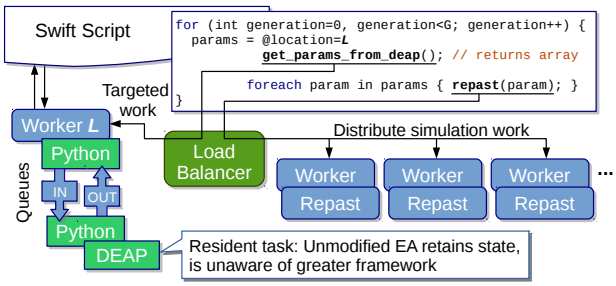

Figure 2: MRTC ensemble framework in Swift/T.

More important, we can more easily leverage third party libraries as resident programs without extensively modifying them to fit a data serialization based scheme.

B. Resident tasks for ensemble control

Previous uses of workflow languages to control parameter searches typically take one of two approaches. In the first approach, the search is encoded in the workflow language. While some workflow languages provide rich support for arithmetic operations (Swift/T is notable in this regard), many do not. Even so, this approach requires that such algorithms be coded from scratch in the workflow language, and it may be impossible to reuse code in other languages. In the second approach, the algorithm is provided as a built-in feature of the workflow system. This approach has been taken by Nimrod/O [28], among others. It does not allow the end users much control over the search algorithm used, unless they can modify the source code of the workflow system itself.

This work defines and uses *resident tasks* as a building block to implement user-defined workflow parameter searches. The key technological feature is the ability to launch a task in a *background* process or thread. *Background* indicates that the *foreground* process or thread returns control to Swift/T after execution (as a normal task would), but the background task is still running. It retains state and potentially performs ongoing computation. For the current example, the background task maintains the state of the EA. The overarching workflow must simply *query* this task for instruction on what tasks to execute next. To do so, a task is issued to the same location as the resident task, which communicates with it over IPC.

C. Queue-based task IPC

To query the state of the EA, we designate one worker on location L for exclusive use by DEAP. Interaction with this worker is shown in Figure 2. The tasks running on this worker use the embedded Swift/T Python interpreter [29], internally controlled by Swift/T through its C interface, to instantiate a Python subthread connected to the parent Python process with two queues, IN and OUT. The Python task then returns control to Swift/T but does not deallocate the Python interpreter at the C level. Thus, when subsequent Python tasks execute on location L, they have access to IN and OUT. The Swift function `get_params_from_deap()`, for example, reads from OUT and returns the DEAP-generated parameter sets

to the Swift level. The user Swift script can iterate over these parameters, passing them to Repast tasks (`repast(param)`) that are load balanced across workers.

We note how IPC is used in this model. Swift/T processes (workers and the load balancer) communicate over MPI and thus may be distributed across nodes. This communication is automated by the Swift/T system to implement the Swift script. Communication on worker L between the embedded Python interpreter and its Python subthread is performed over Python Queues.

D. Evolutionary algorithms and ABM

The adaptive parameter search algorithm we used was a simple EA, built by using the DEAP toolkit. The EA creates an initial population p_0 of parameter combinations, or individuals, of size `pop` and proceeds to iteratively generate populations p_i, for each generation i, based on the evaluated performance of individuals in the previous population p_{i-1}. Each new population p_i is created by first selecting with replacement `pop` individuals from p_{i-1} using a stochastic selection method (tournament selection with size 3 here). This biases p_i to higher performing individuals. Then the population members are mated and mutated to create a new generation of `pop` offspring, where both mating and mutation probabilities are set through user defined parameters. A useful EA in epidemiology, for example, could help select parameters for a contagious disease response strategy.

In this example, we adapted the simple JZombies demonstration model distributed with Repast Simphony. The JZombies model involves two agent types, Zombies and Humans, where the Zombies chase after Humans, seeking to infect the Humans. Once a Human agent is infected it is (monotonically) transformed into a Zombie agent. Thus, each simulation run will eventually see all agents become Zombies. The parameters in the model are the integer type parameters `zombie_count` and `human_count`, the initial number of Zombies and Humans, respectively. To this base model we introduced a varying step size for each of the agent types. The original model had Zombies move in steps of length 1 (in units of the model space) and Humans in steps of length 2. The present model encapsulates these two values into two float type parameters `zombieStep` and `humanStep`. Thus, each EA population member is a tuple (`zombie_count`, `human_count`, `zombieStep`, `humanStep`).

The fitness metric in this example was the remaining number of Humans at a specific simulation time tick, where more surviving Humans corresponds to higher fitness. The remaining Humans across the stochastic variations for each parameter combination were averaged to yield the final fitness value. Figure 3 shows the agent mobility results for a run with `pop` $= 800$. The results confirm the intuition that in a highly fit model, Humans must move as quickly as possible and Zombies must shamble as slowly as possible.

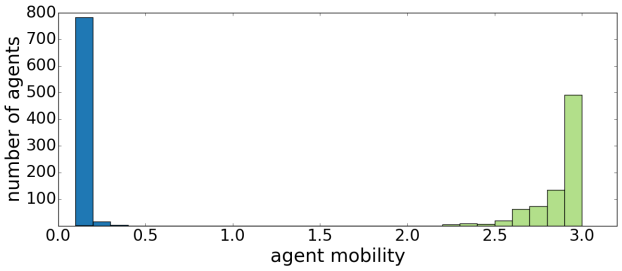

Figure 3: Mobility (`zombieStep` and `humanStep`) histogram for Zombie (blue) and Human (green) agents after 100 generations with an EA using a population of 800.

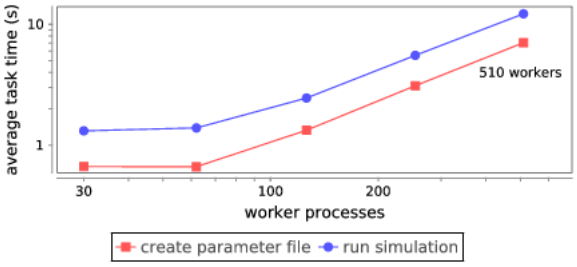

Figure 4: Average task runtime as function of concurrency.

IV. Performance Results

For our performance evaluation, we constructed a test case with the JZombies model. All tests were performed on the cluster *Midway* at the University of Chicago on the "sandyb" partition using Intel Sandybridge 16x cores running at 2.6 GHz with 32 GB memory per node.

This case stresses our system well because it uses simulation tasks that are much shorter than realistic ABMs. Task runtime averages are shown in Figure 4. A "create parameter file" task is used to create stochastic variations of the parameters obtained from the DEAP routine, and these become ABM inputs. Then the "run simulation" task actually runs the simulation. Both involve launching a Java virtual machine. Both take the nominal amount of time, 0.6 seconds and 1.3 seconds, respectively, up to 64 processes. After that, file system contention adversely affects the task runtime. Although mitigation strategies could be used, we did not apply any complex techniques in order to avoid complicating the results of interest to this work.

A. Evolutionary algorithm behavior

As described in §III-D, EAs run a generation of simulations, then spend some time determining what to do next. In this measurement, we treat any time spent outside the ABM as overhead, and we measure the "load" as the fraction of the total computing capability being used at a given point in time. This is illustrated in Figure 5, which shows the load over time. The system alternates between a compute intensive phase, in which the ABMs are computed on the workers, and a low-load phase, in which the system runs out of work to do before the EA produces additional work.

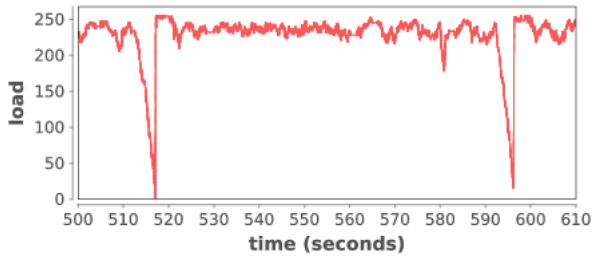

Figure 5: System load over time for 254-worker run.

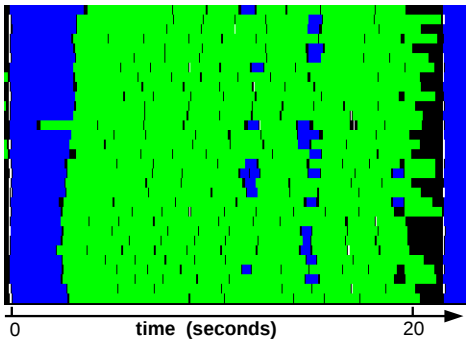

Figure 6: Gantt chart of workflow states for 30-worker run.

The dips in load are investigated by producing a Gantt chart of process states over time. An EA generation is captured from a trace in Figure 6. Each worker process is shown as a row, transitioning from state to state. Blue indicates the "create parameter file" task, green indicates the "run simulation" task, and black is idle time. As shown, the generation begins with a flurry of parameter file creation. Computation dominates the time, except at the end of the generation, when the load balancer runs out of work and an effective barrier waits until all tasks are complete before calling into DEAP to obtain new parameters. The time spent in DEAP is short enough to be invisible on the chart.

To evaluate the scalability of our framework, we measure the average load over the whole run and divide by the hardware concurrency (number of cores), producing a utilization statistic. For increasing concurrency and EA population size, we plot the utilization in Figure 7. As shown, system utilization is good; our maximum measurement on 510 worker processes is 88.95%.

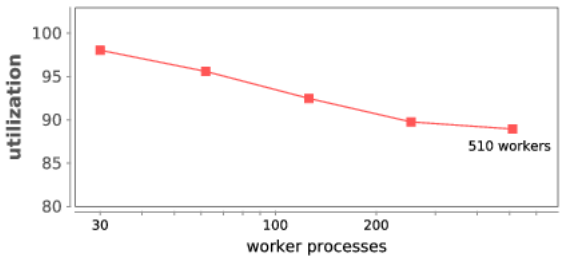

Figure 7: Utilization over increasing run size.

V. Summary

In this paper we have presented a general mechanism for running large ensembles of simulations in which sophisticated statistical algorithms can iteratively and adaptively refine simulation parameters through the analysis of recently generated results and launch new simulations based on the refined parameters. The mechanism itself has been implemented by using the Swift/T dataflow language and exhibits a novel form of inversion of control using location-aware many task scheduling, resident tasks, and non-trivial IPC over HPC resources. Using this framework, third-party adaptive parameter search techniques can be used to quickly and efficiently find global optima, characterize other important regions of parameter spaces, generate distributions of parameter values consistent with observations, and allow the periodic assimilation of real-world observations, to name a few examples. Performance results from this reference implementation illustrate the basic scalability of the framework on a typical cluster.

We intend to develop additional use case examples that exploit widely available statistical libraries, and to release our core features as an extensible framework for the community. Our aim is to make large-scale computing resources far more usable to more researchers in many more science domains by providing easy-to-use, integrated ensemble control solutions. Additionally, we are generalizing the approach to support R libraries for similar IoC capabilities, opening up a greater share of the active open source statistical library resources. Repast HPC [30] is a high-performance computing agent-based modeling toolkit based on C++ and MPI. We are developing IoC workflows that will exploit both the speed and parallelism of distributed Repast HPC ABMs, allowing us to run ensembles on extreme-scale computers.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357; by NSF awards ACI 1148443, BCS 1114851, DEB 1516428; and by NIH award R01AG047869. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility. This work was completed in part with resources provided by the University of Chicago Research Computing Center.

References

[1] T. C. Germann, K. Kadau, I. M. Longini, and C. A. Macken, "Mitigation strategies for pandemic influenza in the United States," *Proceedings of the National Academy of Sciences*, vol. 103, pp. 5935–5940, Apr. 2006.

[2] C. M. Macal, M. J. North, N. Collier, V. M. Dukic, D. T. Wegener, M. Z. David, R. S. Daum, P. Schumm, J. A. Evans, J. R. Wilder, L. G. Miller, S. J. Eells, and D. S. Lauderdale, "Modeling the transmission of community-associated methicillin-resistant Staphylococcus aureus: a dynamic agent-based simulation," *Journal of Translational Medicine*, vol. 12, p. 124, May 2014.

[3] M. Riddle, C. M. Macal, G. Conzelmann, T. E. Combs, D. Bauer, and F. Fields, "Global critical materials markets: An agent-based modeling approach," *Resources Policy*, vol. 45, pp. 307–321, Sept. 2015.

[4] J. Ozik, N. Collier, J. T. Murphy, M. Altaweel, R. B. Lammers, A. A. Prusevich, A. Kliskey, and L. Alessa, "Simulating water, individuals, and management using a coupled and distributed approach," in *Proceedings of the 2014 Winter Simulation Conference*, WSC '14, (Piscataway, NJ, USA), pp. 1120–1131, IEEE Press, 2014.

[5] F. Bert, M. North, S. Rovere, E. Tatara, C. Macal, and G. Podest, "Simulating agricultural land rental markets by combining agent-based models with traditional economics concepts: The case of the Argentine Pampas," *Environmental Modelling & Software*, vol. 71, pp. 97 – 110, 2015.

[6] J. M. Wozniak, T. G. Armstrong, M. Wilde, D. S. Katz, E. Lusk, and I. T. Foster, "Swift/T: Scalable data flow programming for distributed-memory task-parallel applications," in *Proc. CCGrid*, 2013.

[7] F.-A. Fortin, F.-M. D. Rainville, M.-A. Gardner, M. Parizeau, and C. Gagn, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, July 2012.

[8] M. J. North and C. M. Macal, *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation*. Oxford University Press, 1 ed., March 2007.

[9] J. Shaman and A. Karspeck, "Forecasting seasonal outbreaks of influenza," *Proceedings of the National Academy of Sciences*, vol. 109, pp. 20425–20430, Dec. 2012.

[10] J. Shaman, A. Karspeck, W. Yang, J. Tamerius, and M. Lipsitch, "Real-time influenza forecasts during the 2012-2013 season," *Nature Communications*, vol. 4, Dec. 2013.

[11] W. Yang, A. Karspeck, and J. Shaman, "Comparison of filtering methods for the modeling and retrospective forecasting of influenza epidemics," *PLoS Comput Biol*, vol. 10, p. e1003583, Apr. 2014.

[12] J. Shaman, W. Yang, and S. Kandula, "Inference and Forecast of the current West African Ebola outbreak in Guinea, Sierra Leone and Liberia," *PLoS Currents*, 2014.

[13] J. M. Epstein, "Modelling to contain pandemics," *Nature*, vol. 460, p. 687, Aug. 2009.

[14] J. C. Thiele, W. Kurth, and V. Grimm, "Facilitating parameter estimation and sensitivity analysis of agent-based models: A cookbook using NetLogo and R," *Journal of Artificial Societies and Social Simulation*, vol. 17, no. 3, p. 11, 2014.

[15] G. E. P. Box, J. S. Hunter, and W. G. Hunter, *Statistics for Experimenters: Design, Innovation, and Discovery*. Hoboken, N.J: Wiley-Interscience, 2nd edition ed., May 2005.

[16] M. D. McKay, R. J. Beckman, and W. J. Conover, "Comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, pp. 239–245, May 1979.

[17] M. D. Morris, "Factorial sampling plans for preliminary computational experiments," *Technometrics*, vol. 33, pp. 161–174, May 1991.

[18] S. Kirkpatrick, "Optimization by simulated annealing: Quantitative studies," *Journal of Statistical Physics*, vol. 34, pp. 975–986, Mar. 1984.

[19] R. Verfrth, "A posteriori error estimation and adaptive mesh-refinement techniques," *Journal of Computational and Applied Mathematics*, vol. 50, no. 13, pp. 67 – 83, 1994.

[20] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Cambridge, Mass: A Bradford Book, Apr. 1992.

[21] M. A. Beaumont, "Approximate Bayesian computation in evolution and ecology," *Annual Review of Ecology, Evolution, and Systematics*, vol. 41, no. 1, pp. 379–406, 2010.

[22] F. Hartig, J. M. Calabrese, B. Reineking, T. Wiegand, and A. Huth, "Statistical inference for stochastic simulation models theory and application," *Ecology Letters*, vol. 14, pp. 816–827, Aug. 2011.

[23] G. Evensen, *Data Assimilation - The Ensemble Kalman Filter*. Springer-Verlag Berlin Heidelberg, 2nd ed., 2009.

[24] N. Gordon, D. Salmond, and A. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *IEE Proceedings F Radar and Signal Processing*, vol. 140, no. 2, p. 107, 1993.

[25] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, pp. 174–188, Feb. 2002.

[26] M. J. North, N. T. Collier, J. Ozik, E. R. Tatara, C. M. Macal, M. Bragen, and P. Sydelko, "Complex adaptive systems modeling with Repast Simphony," *Complex Adaptive Systems Modeling*, vol. 1, p. 3, March 2013.

[27] F. R. Duro, J. G. Blas, F. Isaila, J. Carretero, J. M. Wozniak, and R. Ross, "Exploiting data locality in Swift/T workflows using Hercules," in *Proc. NESUS Workshop*, 2014.

[28] D. Abramson, A. Lewis, T. Peachey, and C. Fletcher, "An automatic design optimization tool and its application to computational fluid dynamics," in *Proc. SuperComputing*, 2001.

[29] J. M. Wozniak, T. G. Armstrong, K. C. Maheshwari, D. S. Katz, M. Wilde, and I. T. Foster, "Toward interlanguage parallel scripting for distributed-memory scientific computing," in *Proc. CLUSTER*, 2015.

[30] N. Collier and M. North, "Parallel agent-based simulation with Repast for High Performance Computing," *SIMULATION*, Nov. 2012.