Using Distributed Analytics to Enable Real-Time Exploration of Discrete Event Simulations

Matthew Malensek*, Walid Budgaga*, Sangmi Pallickara*, Neil Harvey†, F. Jay Breidt‡, and Shrideep Pallickara*
*Department of Computer Science, ‡Department of Statistics
Colorado State University, Fort Collins, Colorado, USA
Email: *{malensek, budgaga, sangmi, shrideep}@cs.colostate.edu, ‡jbreidt@stat.colostate.edu
†School of Computer Science
University of Guelph, Guelph, ON, Canada
Email: neilharvey@gmail.com

***Abstract*—Discrete event simulations (DES) provide a powerful means for modeling complex systems and analyzing their behavior. DES capture all possible interactions between the entities they manage, which makes them highly *expressive* but also compute-intensive. These computational requirements often impose limitations on the breadth and/or depth of research that can be conducted with a discrete event simulation.**

This work describes our approach for leveraging the vast quantity of computing and storage resources available in both private organizations and public clouds to enable real-time exploration of a discrete event simulation. Rather than considering the execution speed of a single simulation run, we autonomously generate novel *scenario variants* to explore an entire subset of the simulation parameter space. These workloads are orchestrated in a distributed fashion across a wide range of commodity hardware. The resulting outputs are analyzed to produce models that accurately forecast simulation outcomes in real time, providing interactive feedback and bolstering research possibilities.

***Index Terms*—Discrete Event Simulation, Latin Hypercube Sampling, Distributed Execution, Cloud Infrastructure**

I. Introduction

The behavior of complex, real-world systems is often difficult to predict or fully understand. These systems may be influenced by any number of internal or external stimuli, and direct experimentation is often prohibitively expensive, time-consuming, or simply not feasible. In these situations, computer simulation is a compelling solution. Specifically, discrete event simulations (DES) model all possible interactions between entities in a system, making them highly *expressive*. To model uncertainty in these interactions, *stochastic* discrete event simulations associate probabilities with their events. However, this expressiveness comes at the cost of increased computational complexity and prolonged execution times.

Our subject discrete event simulation, NAADSM, [1] is an epidemiological model of disease outbreaks in livestock populations. It has been applied in studies of several different diseases, including foot-and-mouth disease [2], avian influenza [3], and pseudorabies [4]. NAADSM is a stochastic DES: simulations are run many times, with each *iteration* contributing to an overall representation of the output variables' *probability distributions*. Iterations often require several hours of CPU time to execute depending on how events unfold.

The computational complexity of these stochastic iterations makes it difficult for planners and epidemiologists to perform exploratory "what if" analysis that play an important role in planning and preparedness. For instance, a planner may make subtle adjustments to quarantine procedures or the number of vaccines available in order to analyze economic consequences or changes in disease spread. Each change to the input parameters requires a new set of iterations to be run. Dividing the target simulation into several units and executing them in parallel is a possible solution to this problem [5], [6], but generally does not enable real-time exploration.

This paper describes our approach for retaining the expressiveness of stochastic DES while addressing the weaknesses in the timeliness of their outcomes. We achieve this by utilizing voluminous epidemic simulation data to glean insights and derive relationships between scenarios and outcomes. We then use this information to create models that can forecast the results for an entire class of input parameters, enabling our system to provide real-time answers to exploratory investigations.

A. Research Questions

We consider the problem of generating fast, accurate DES forecasts for a given subset of the input parameter space. These forecasts are generated in lieu of compute-intensive simulation runs. This objective provides several research challenges, such as managing input data types and their respective bounds, ensuring interactivity, and providing accurate predictions. Specific research questions we explore include:

1) How can we minimize the number of iterations required to build our models while still ensuring statistical coverage of the parameter space?
2) What are the implications of our execution model, and how can we obtain necessary processing resources?
3) A large amount of training data is necessary for making predictions. How can this data be managed in a scalable and fault-tolerant manner?
4) How can we deal with increases in dimensionality as the number of input parameters grows?
5) What prediction models can provide both **accurate** and **real-time** results?

B. Summary of Approach

Our approach treats the DES in question as a black box and focuses on deriving relationships between the inputs and outputs. Given a disease spread scenario, our framework views input tuples as points in the multidimensional parameter space. We first derive bounds for each of the dimensions from both historical data and subject-matter experts, and then sample within this parameter space to create novel *scenario variants*. Our objective is two-fold: we wish to ensure adequate coverage of the parameter space, while also controlling the size of computational workloads.

For each scenario, we inspect the variances of key output variables to derive the number of iterations that must be executed. Both the variant generation and their subsequent simulation iterations are implemented as MapReduce [7] jobs that are orchestrated by our *Forager* component. Forager deals with highly-elastic resource pools and can scavenge for CPU cycles on both physical and virtual machines, including spot instances. These simulation runs generate a large amount of data, often producing over 1 TB of outputs in a few hours. To cope with these storage demands, we use a distributed storage system to manage the data in a fault-tolerant manner.

Once the simulation iterations have been executed, we model the relationships between inputs and outputs. To facilitate predictions, we create a model for each output variable. We consider both linear (multivariate linear regression) and non-linear (artificial neural networks) methods to construct these models. The entire process enables our system to provide accurate answers to "what if" scenarios in real time.

C. Paper Contributions

This paper describes our approach for supporting interactive exploration of discrete event simulations. The research highlights the use of analytics to ensure accurate, timely forecasts that account for: statistical coverage of the parameter space, orchestration of workloads, generation and management of training data, correlations between inputs and outputs, dimensionality reduction, and the use of learning structures. Our framework is broadly applicable to other discrete event simulations, including those that deal with high dimensionality. We also offer a flexible orchestration component and manage the resulting datasets with a distributed hash table based key-value store. Finally, we have demonstrated the use of both linear and non-linear predictions in facilitating real-time exploration.

D. Paper Organization

The rest of the paper is organized as follows. Section II describes our scenario variant generation process. Section III focuses on our distributed execution platform, Forager. Section IV outlines how we manage outputs and distributed state, followed by Section V, which describes how we build our models and make predictions. Section VI surveys related work, and Section VII provides concluding material and our future research direction.

II. Generating Novel Simulation Variants

In our subject simulation, input parameters are used to describe disease properties and outbreak characteristics. These variables include factors such as the probability of infection transfer, maximum airborne distance of disease spread, and the overall area at risk for infection. The first piece of information required to generate a new scenario is the *data type* for each input variable, which could include booleans, integers, floating point values, or even line charts and probability density functions (PDFs). Our framework can identify most data types automatically, but for more exotic parameters we provide an XML-based variable description language.

While some input variables have predefined ranges of valid values (such as a percentage ranging from 0 to 100%), others are completely unconstrained. In both cases, a value may be **valid** but not **plausible**, i.e., extremely unlikely to occur due to environmental conditions or other factors. To produce plausible ranges for the input variables, our framework consults historical data available in previous scenarios as a preliminary step; for instance, Figure 1 contains a variety of probability density functions that were used previously for the "cattle latent period" input parameter (amount of time between an infection and the onset of infectiousness, in days). Next, the ranges determined by this process are inspected and refined by subject-matter experts if necessary. This process helps reduce or potentially avoid user intervention while maintaining accuracy.

A. Complex Data Types: Charts and Probability Densities

While most input variables represent a numerical value or discrete state, simulations also frequently employ two-dimensional line charts or probability density functions to describe complex behavior. These data types play a vital role in simulation outcomes and must be varied to enable the exploration of a scenario's parameter space. However, a simple range of values does not capture the multidimensional relationships that these data types describe.

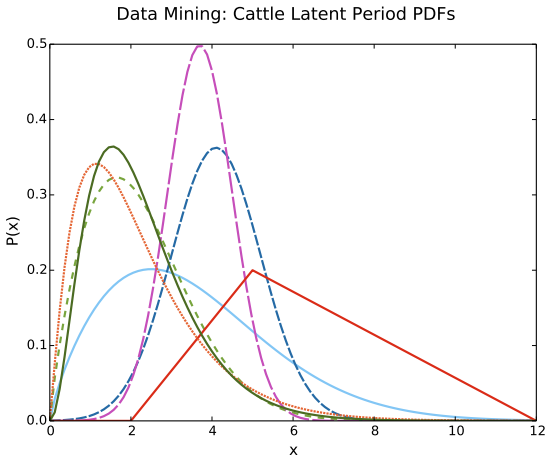

Fig. 1. A variety of probability density functions (PDFs) that were used to describe the "cattle latent period" parameter in previous scenarios.

To generate a 2D line chart variant, we consider four variables that describe chart behavior: the span of x- and y-values, maximum x- and y-values, x-value at the maximum y-value, and the percent distance across the x-axis where the maximum y-value occurs. Next, we perturb the data points to create a new chart that exhibits behavior similar to the source chart, while still representing the shift in values derived from historical trends and feedback from subject-matter experts.

Probability density functions can often be decomposed into a few distinguishing variables as well: mean, variance, and skewness can all be manipulated to create new PDF variations. Unfortunately, each type of distribution has its own formulas for modifying these attributes, and the fact that our simulation supports 22 different types of distributions only further complicates matters. Instead of dealing with this issue as 22 separate problems, (or possibly more for other simulations) we generalize the PDFs by transforming them to *piecewise linear approximations*. Once this step has been completed, we inspect the resulting upper and lower bounds, mean, variance, and skewness. These attributes are modified to create a new linear approximation of a curve. Next, a *beta distribution* is mathematically fit to the curve. Beta distributions are described by two *shape parameters*, α and β, which can be adjusted to model a wide range of distributions: normal, continuous, skew normal, exponential, etc. An overview of our PDF generation algorithm is provided in Figure 2. We have verified that this approach works with the 22 PDFs supported by NAADSM, including Bernoulli, hypergeometric, binomial, and logistic distributions.

B. Latin Hypercube Sampling

After establishing plausible input ranges, one might elect to generate new scenario variants with random samples across the parameter space. However, simple random sampling assigns an equal probability to each possible input without considering the plausibility of the values. To circumvent this limitation, Monte Carlo sampling methods draw values from a probability distribution. This preserves the plausibility curve of potential inputs, but also increases the likelihood of choosing highly probable values; if a small number of samples are drawn from the probability distribution, a correspondingly small portion of the input space will be represented. This means that we would have to generate (and execute) a large number of scenario variants to adequately explore the parameter space with Monte Carlo sampling.

Unlike simple random sampling or Monte Carlo sampling methods, Latin Hypercube Sampling (LHS) [8] *stratifies* the input probability distributions to better represent their underlying variability. This reduces the number of samples required for our algorithms to adequately explore the scenario parameter space, which in turn decreases the overall computational footprint of the framework. LHS owes its name to *Latin squares*, which are $N \times N$ arrays that contain N different elements. Each element in a Latin square occurs exactly once in each row and column. When this concept is applied in a multidimensional setting the elements occur once in each hyperplane, forming a *Latin hypercube*. This allows us to produce samples across all the variables in the parameter space in a single sampling step. Based on the number of samples required, each stratum represented by the elements in the hypercube is divided into equal intervals. These produce samples in the range $[0..1]$, which are converted back to the original units using the information from our data mining process.

C. Measuring and Verifying Output Variance

Once a scenario variant has been created, it must be executed several times to obtain an understanding of its output distribution and behavior. To begin this process, 32 pilot runs are executed for each variant. After the pilot runs are complete, our framework determines whether the overall variation of the output variables is of *practical significance* or not. Doing so requires two pieces of information: (1) the outputs that are most meaningful from an analytical standpoint, and (2) the *minimal significant difference* in output variances that must be achieved. As with input range discovery, both of these items are obtained from mining historical data and subject-matter experts.

To establish a set of variables that have a strong analytical significance, our framework inspects reports generated from past scenario executions to determine which variables were requested most frequently by end users. This process can be supervised by subject-matter experts or performed autonomously. Once the prediction engine has been bootstrapped, our framework can also make recommendations on which output variables might warrant further analysis.

Determining the minimal significant difference in output variables requires the knowledge of a subject-matter expert. These values can be expressed as percentages, numeric ranges, or confidence intervals, and tell the framework whether or not there is enough variation in the output variables. They also determine how well the input parameter space has been explored based on the overall variance of the outputs. If the minimum variance is met, execution of the scenario variant is complete. Otherwise, the observed variance is used to calculate how many more executions of the scenario must be carried out to achieve the required minimum variations.

III. Distributed Simulation Orchestration: Forager

Our framework requires a large number of processing resources, with each scenario produced by the simulation variant generator representing several discrete units of computation. In our initial tests, 10,000 variants were generated and each was run for at least 32 iterations. After running additional simulations to achieve target output variances (as discussed in the previous section), the total number of iterations reached approximately 400,000. This makes our framework an ideal candidate for execution in an elastic cloud or clustered environment due to its computational footprint and the uncertainty in total iterations required. Compared to the problems addressed by distributed execution frameworks

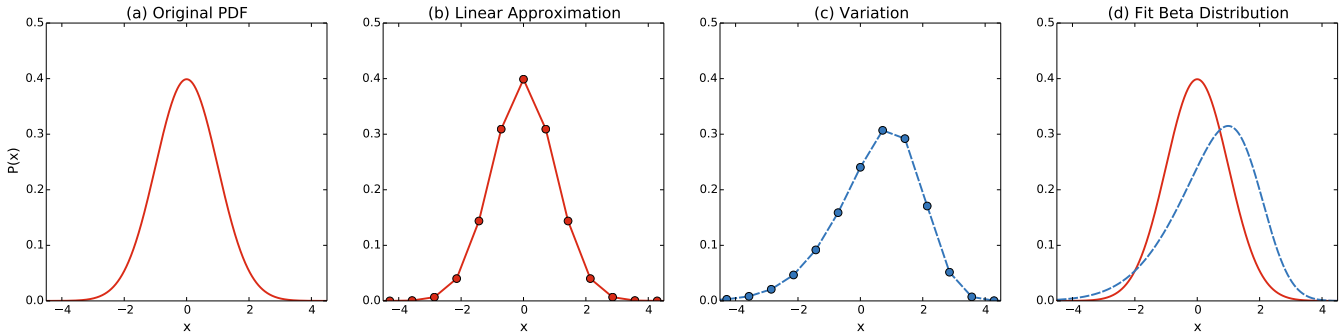

Fig. 2. A visual overview of our probability density function (PDF) variation algorithm. The original PDF **(a)** is converted to a piecewise linear approximation **(b)**. Next, key attributes (upper and lower bounds, mean, variance, and skewness) are modified to create a linear approximation of a new PDF **(c)**. Finally, a beta distribution is fit to the modified linear approximation to create a new PDF variant **(d)**.

such as Hadoop [9], Dryad [10], or the myriad of other MapReduce [7] implementations, the tasks we execute and manage have several distinct features:

- Run times are uncertain, and vary due to the stochastic nature of the underlying simulation.
- The overall number of tasks is significantly larger than the number of available processing resources.
- Tasks can be completed out-of-order, and there are no "waves" of execution.
- The pool of processing resources is highly elastic, fluctuating constantly as availability changes over time.
- The framework must deal with other users and processes contending for resources.

Because of these processing requirements, we chose to design a new distributed execution engine, called *Forager*. Similar to animal behavior observed in nature, Forager must adapt to constantly changing external conditions to acquire resources. Forager is based on our *Granules cloud runtime* [11], and provides new scheduling and orchestration functionality for our specific use case.

A. Resource Acquisition

For a given scenario, our framework may produce hundreds of thousands of executable tasks. This creates a large demand for processing elements, which Forager acquires from a variety of sources: clusters, idle workstations, and both public and private clouds. Unlike volunteer computing [12] deployments, typical Forager installations are managed by a single entity in a trusted environment. This constraint helps us ensure that confidential or proprietary information being used by the simulation is not made publicly available.

A lightweight Forager *daemon* is run on each participating resource. Rather than being managed by a central server or a coordinating node, the daemons securely connect to a distributed file system that maintains a set of pending tasks and *processing directives*. These processing directives allow the administrator(s) of the Forager cluster to assign specific rules to the resources. Directives include items such as the particular time of day that the resource may be used, the maximum number of cores that should be assigned to tasks, process priorities, and requirements for specific hardware. When the processing directives permit, the daemon will remove one or more of the tasks from the *pending task queue* and begin execution.

B. Task Composition

Forager tasks are composed of several processing steps. Scenario variants are created in an initial partitioning phase and stored in the distributed file system. Next, the variants are loaded and executed during the *map* phase. In the *reduce* phase that follows, raw simulation outputs are compressed and filtered to produce a final dataset that is used for knowledge extraction.

Since our MapReduce implementation must deal with frequently changing execution conditions, it provides three options for stopping a running task: immediate termination, memory-resident suspension, and hibernation. *Immediate termination* results in the loss of all progress made on the task, but releases all resources immediately. This feature is useful in situations where the host machine must shift resources at a moment's notice. *Memory-resident suspension*, on the other hand, ceases execution but keeps the simulation in memory, which is helpful when an idle resource becomes busy for a short period of time. Finally, we added an optional *hibernation* function that Forager tasks can implement to gracefully serialize their state and store it in the distributed file system. This allows Forager to cope with situations where a task needs to be migrated to a different machine or when processing directives will prohibit execution for a substantial amount of time. Hibernating a task requires some time to complete; for our particular simulation, tasks took about 4.2 seconds to hibernate on average (over 1000 samples).

C. Situational Scheduling

Diverging from the standard MapReduce execution model, Forager daemons *pull* tasks from the distributed task queue when they can contribute CPU cycles or other resources. This allows dedicated hardware to continually execute new tasks, while daemons on shared systems can wait for free resources or until processing directives are met. To facilitate this approach, the Forager daemon monitors performance statistics

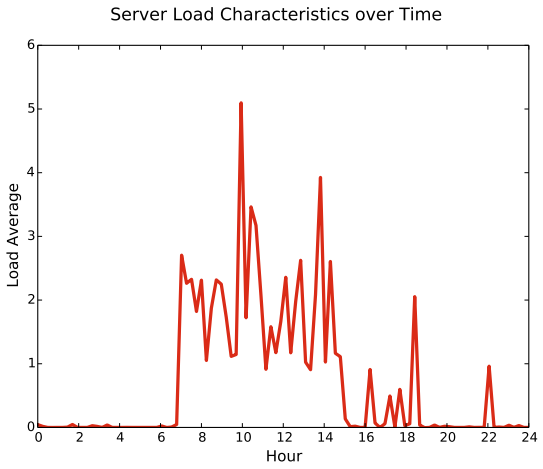

Fig. 3. One-minute load average on a busy 4-core web server. Note the increase in peak load during working hours (7 AM to about 3 PM). At night, the server is mostly idle.

on a per-machine basis. These statistics include the CPU idle time, steal time (in virtualized environments), memory and disk usage, load averages, the context switch rate, and the current number of active users on the machine.

While a scenario variant that has been running much longer than usual may simply represent a particularly CPU-intensive chain of events unfolding in the simulation, it can also indicate that the daemon managing the task has insufficient resources. We use the aforementioned performance statistics to track resource utilization, and generate *speculative tasks* in the event of a slowdown. Speculative tasks in our framework have the option of using previously-hibernated state information to help reduce duplicate processing work, unlike the speculative tasks seen in frameworks such as Hadoop [9].

Idling workstations and servers were one of the primary motivating factors cited by Tanenbaum [13] for distributed computing. These idle resources are especially prevalent in large businesses and academic settings, and are prime targets for *cycle scavenging* (exploiting unused processing resources). Figure 3 demonstrates this phenomenon on a busy web server at Colorado State University: during working hours (from about 7 AM to 3 PM) the load average (number processes waiting for or using the CPU) remains fairly high. However, during nighttime the server is mostly idle, presenting an opportunity for our framework to run scenario variants. Forager conducts *situational scheduling* to exploit these occurrences, where task scheduling is based on current machine load characteristics and the time of day (dervied from temporal usage patterns or preset intervals).

D. Leveraging Elastic Clouds

For situations where the processing requirements of a simulation outstrip the capabilities of the resources acquired through cycle scavenging and private clusters, we incorporated support for cloud deployments as well. Forager allows participating hardware to join and leave the system at any time, making elastic cloud services an ideal means to supplement its resource pool. Furthermore, Amazon provides EC2 *spot instances*, which enable users to trade reliability for lower prices. In essence, spot instances apply the laws of supply and demand to virtualization: users issue a *spot request* with their desired virtual machine (VM) characteristics, a maximum price, and the dates and times the request is valid for. If the market price of the VM exceeds the maximum specified, the spot instance is terminated. On average, spot instances cost 50-60% less than their traditional counterparts.

In our use case, reliability is a secondary concern behind overall processing throughput. Tasks are lightweight, self-contained, and are executed over a relatively short period of time. However, the spot VMs we used were never terminated during our testing period (likely due to running during low-demand summer months). We tested a range of VMs on the Amazon public cloud as well as our own private cloud. Table I illustrates the performance differences between the configurations. These include VMs from the Amazon m1, c1, m3, c3, and t2 instance families, along with virtualized and bare metal results for our hosts running the KVM hypervisor on Fedora 20 (Xeon E5620 with 12 GB RAM, Xeon E3-1220v2 with 8 GB RAM).

TABLE I
SCENARIO EXECUTION TIMES IN A VIRTUALIZED ENVIRONMENT AVERAGED OVER 1000 ITERATIONS.

Hardware	Execution Time (s)	Execution SD (s)
t2.micro	442.94	215.79
m3.medium	118.84	21.04
c3.large	57.07	10.06
m1.medium	51.80	8.87
c1.xlarge	50.79	8.82
c1.medium	48.66	7.03
E5620 (KVM)	64.90	10.47
E5620 (Bare)	60.02	8.60
E3-1220v2 (KVM)	49.86	7.76
E3-1220v2 (Bare)	47.33	5.70

While these results closely mirror the theoretical performance differences between instances, it is worth noting that the t2.micro can also achieve competitive results when enough CPU credits are available for "burst" performance, which temporarily allows the instance to consume more than its baseline allotted CPU time. When burstable, the average scenario execution time on a t2.micro was 99.27 seconds. Since Forager can migrate longer-running tasks and monitor CPU steal time at each resource, burstable instances are a viable option for cycle scavenging as long as the primary function of the VM does not tax the CPU. Interestingly, the previous-generation m1 and c1 instances exhibited better performance than their newer counterparts for our particular workload. In general, we chose the lowest priced VMs and avoided m3.medium instances unless there was a substantial cost benefit. Our

systems experienced 5.3% and 8.1% virtualization overheads on the E5620 and E3-1220v2 processors, respectively.

IV. Output Management and Storage

Rather than relying on a central server or coordinator process, our framework stores its persistent state in a distributed file system. This information includes pending and executing tasks, resource performance statistics, and the overall system status. Even more importantly, the distributed file system is tasked with managing and storing simulation outputs as well. These outputs must be stored in a scalable and fault-tolerant manner: the 400,000 iterations produced from our single pilot scenario (one in a multitude of exploration possibilities) consumed about **1 TB** of storage space. We use our Galileo [14], [15] DHT-based key-value store to fulfill these requirements. Galileo is a distributed, fault-tolerant, and document-oriented storage system, making it an ideal candidate for managing the JSON output files produced by our subject simulation.

A. Output Compression

To help manage output file sizes, we evaluated each of the compression algorithms available in Galileo: LZO, DEFLATE, Burrows-Wheeler, and LZMA. Outputs were stored in append-only *blocks* of approximately 1,000 MB each (320 simulation iterations) before compression. Table II contains the resulting file sizes and their corresponding compression ratios for each of the algorithms surveyed.

TABLE II
Output compression averaged over 32,000 iterations.

Algorithm	File Size (MB)	Compression Ratio
No Compression	978.2	1.0
LZO	174.1	5.6
DEFLATE	97.2	10.0
Burrows-Wheeler	37.6	26.0
LZMA	34.1	28.6

For the JSON outputs produced by our simulation, the LZMA algorithm provided the best average compression ratios. However, we also considered compression speed; Table III provides compression and decompression times for each of the algorithms.

TABLE III
Compression and decompression times averaged over 32,000 iterations.

Algorithm	Compression (s)	Decompression (s)
LZO	1.9	6.4
DEFLATE	16.5	6.6
Burrows-Wheeler	179.0	15.7
LZMA	211.43	7.61

In our specific use case, raw outputs will be compressed once and decompressed several times later during analysis and forecasting. This led us to choose the LZMA algorithm due to the compression ratios it achieved on our dataset as well as its decompression times, which were competitive with the fastest algorithms (LZO and DEFLATE). Ultimately, output compression saves a substantial amount of disk space and greatly increases exploration capabilities on a given set of hardware.

V. Knowledge Extraction: Modeling and Prediction

After orchestrating our scenario variants across the resource pool and storing their outputs, we begin the final processing step of our framework: building predictive models. These models generalize the scenarios to allow interactive exploration of their parameter space. Building the models is a one-time process that bootstraps our real-time forecasting engine. To make the predictions, we used both multivariate linear regression and artificial neural networks (ANNs).

A. Dimensionality Reduction

Our particular simulation involves a large number of inputs and outputs. When making predictions, these values contribute to a very high overall dimensionality. To help reduce the effects of the *curse of dimensionality*, we evaluated two methods commonly used for dimensionality reduction: *principal component analysis* (PCA) and *correlation analysis*. PCA is a method that can be used to project a dataset onto a lower dimensional space. This is achieved by selecting components that contribute the most to the underlying variability in the data. However, PCA does not consider the relationship between input and output variables in our case, which can lead to the removal of some inputs that may influence outcomes. To avoid this issue, we used correlation analysis with the Pearson correlation coefficient to measure the degree of correlation between input variables and the outputs. Using this approach enables us to select input variables that tend to have a strong influence on simulation outcomes, which are then used to build our models.

B. Prediction Methods

To create our validation and test datasets, we used k-fold cross-validation with $k = 10$. We generated our models with artifical neural networks (ANNs) and multivariate linear regression, and then evaluated the root-mean-square error (RMSE), creation times, and prediction times of both methods. In our case, RMSE (the average prediction deviation) and prediction times are critical in evaluating both the accuracy of our forecasts and the overall forecasting speed. For both methods, we generated an individual model for each output variable.

Artificial neural networks are non-linear computational models inspired by the characteristics of biological neural networks. ANNs can be used for a variety of machine learning applications, and are composed of interconnected *neurons* that are responsible for processing information. In our framework,

we used a feedforward neural network trained with backpropagation. The network was configured with one hidden layer and six neurons.

Multivariate linear regression is an approach used for modeling the relationships between multiple dependent variables and multiple independent variables. It produces a set of linear predictor functions that we then use to forecast scenario outcomes. We evaluated several regression options and settled on the Least Absolute Shrinkage and Selection Operator (LASSO) method, which penalizes the absolute size of regression coefficients to help reduce the influence of variables that have little impact on the model. Out of the methods we tested, LASSO provided the best predictive results.

C. Experimental Results

To evaluate our models, we used the inputs and outputs from the 10,000 scenario variants produced by our framework. We considered a total of 1,812 raw input variables, and selected 10 key output parameters based on the guidance we received from epidemiologists. These outputs include items such the disease duration and number of infected animals in the scenario, which are helpful in various forms of analysis (such as determining the economic impact of a particular type of outbreak).

Table IV contains performance statistics for both of our prediction models built with a 133 input parameter set that exhibited high correlation with the output parameters. Predictions were performed 1,000 times to illustrate how the framework would perform in a situation that required results from a large number of models. Multivariate linear regression offered the best performance in both of our timing criteria (time to build the model, and time to make a prediction), but it is worth noting that both methods provided sub-second prediction performance.

TABLE IV
MODEL GENERATION TIMES AND PREDICTION PERFORMANCE.

Model	Build (s)	Predict 1,000 Times (ms)
Regression	1.5	0.2
Neural Network	7998.0	75.0

To visualize the prediction accuracy of our framework, Figure 4 contains the actual values of the disease duration output variable plotted against predictions generated with multivariate linear regression. Values close to the 45-degree reference line are highly accurate. The root-mean-squared error in this test was **4.4 days**, while the RMSE exhibited by our neural network was **5.7 days**. Overall, these values indicate that both models were able to fit the data and provide forecasts in a timely manner, but multivariate linear regression provided better performance in our specific use case.

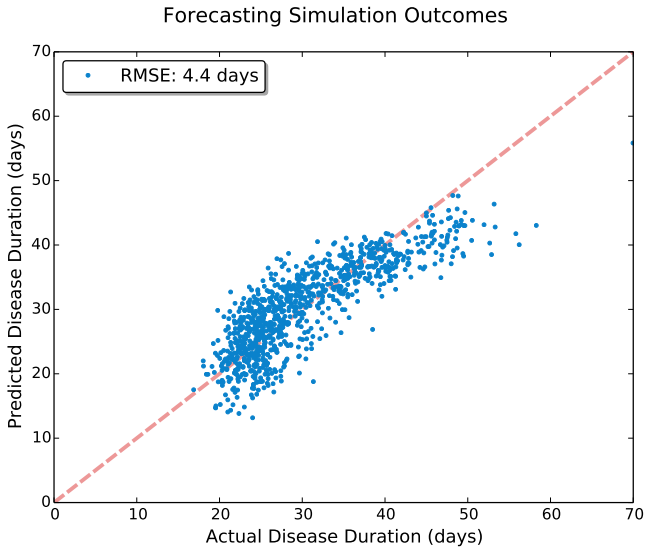

Fig. 4. Prediction accuracy for disease duration using multivariate linear regression. Samples close to the 45-degree reference are highly accurate.

VI. Related Work

Parallel and distributed discrete event simulation has been well-studied in the literature [5], [16], [17]. While these simulations often have numerous parallelization opportunities, they also generally require fine-grained synchronization. Most importantly, a parallel version of a DES must ensure correctness, i.e., the parallelization does not change the output of an identical simulation run on a single thread. Our framework avoids these issues by treating the DES as a black box, and contrasts with parallel DES by providing real-time forecasts. However, parameters discovered with our framework that produce an outcome of scientific interest could also be run in a parallel DES to confirm the results in an efficient manner.

Hadoop [9] and its accompanying file system, HDFS [18] share some common objectives with our framework. Hadoop is an implementation of MapReduce [7], which often involves multiple *waves* of execution that must be completed before the next wave can start. Additionally, the system is often installed on a dedicated cluster and does not need to suspend or migrate tasks when the underlying resources become busy. Like the distributed file system we use in this work, HDFS replicates information across multiple physical machines to ensure failures do not result in data loss.

The Berkeley Open Infrastructure for Network Computing (BOINC) [12] is a volunteer computing platform that enables home users or organizations to contribute their idle processing resources towards a variety of scientific projects. The platform can also be used privately by organizations to create a lightweight grid environment. Unlike our framework, BOINC is generally run on untrusted hardware and requires duplication of tasks to ensure the validity of their outputs. Additionally, BOINC clients are usually deployed on single-user computing devices rather than public resources, and are administered individually.

Grid computing technologies, such as the Globus Toolkit [19] or computing management frameworks such as HTCondor [20] share a common goal of creating distributed processing and storage environments. These deployments often

combine the computing hardware of multiple organizations into a single coherent (and heterogeneous) resource pool. They also support *cycle scavenging*, where idle resources are used for background processing. Unlike volunteer computing, these execution frameworks are administered by a central organization and do not have to deal with untrusted resources. Our framework is designed for single-organization installations, and requires less administrative setup and maintenance.

VII. Conclusions and Future Work

Producing accurate forecasts for discrete event simulations involves: (1) ensuring coverage of the parameter space, (2) efficient orchestration of workloads, (3) amortizing the I/O costs associated with data accesses, (4) coping with dimensionality, (5) building and training lightweight prediction models, and (6) carefully planning which aspects of the framework are in the critical path.

Ensuring statistical coverage of the parameter space provides us with better training data for the learning structures and prediction models. Forager's pull-based approach during orchestration of workloads allows nodes to take on tasks when they are able: lightly used machines execute more tasks than others and a have greater share of the computational workload. Our orchestration scheme works well with both physical machines in shared, public clusters and virtual machines in cloud settings, including spot instances. The use of a DHT-based key-value store, Galileo, allows us to distribute the I/O loads of the outputs generated by the simulation for training of the prediction models.

Dimensionality reduction allows us to identify inputs that contribute to the outputs. Pruning of the input parameter space allows us to better train the prediction models. The pruning process reduces training times and also improves the accuracy of the predictions by eliminating inputs that are sources of statistical noise. Our approach balances the costs associated with training and making predictions. Though the training process is compute-intensive, it is not in the critical path during predictions. Once the training process is complete, making the forecasts based on the constructed models is lightweight and simply involves dot product calculations that can be done in real time. Building a prediction model per output parameter allows identification of inputs (and their respective contributions) to the output. Though this increases the overall training time, the improved accuracy in the predictions offsets this cost.

Our future work will focus on both the orchestration framework and our predictive models. While Forager can exploit spot instances, learning structures or rule-based directives could be used to optimize for the cost of VMs as well. On the prediction front, we will investigate the use of other learning structures such as random forests. We will also evaluate canonical correlation analysis to aid in the reduction of dimensionality.

Acknowledgments

This research has been supported by funding (HSHQDC-13-C-B0018) from the US Department of Homeland Security's Long Range program.

References

[1] N. Harvey, A. Reeves, M. Schoenbaum *et al.*, "The North American Animal Disease Spread Model: A simulation model to assist decision making in evaluating animal disease incursions," *Preventive Veterinary Medicine*, vol. 82, no. 3, pp. 176–197, 2007.

[2] D. Pendell *et al.*, "The economic impacts of a foot-and-mouth disease outbreak: a regional analysis," *Journal of Agricultural and Applied Economics*, vol. 39, no. 0, pp. 19–33, 2007.

[3] C. Green *et al.*, "Simulation modeling of alternative control strategies for an HPAI outbreak using NAADSM," in *Canadian Association of Veterinary Epidemiology Preventive Medicine (CAVEPM) Meeting, May 29 - 30 2010, Guelph, Ontario, Canada*, 2010.

[4] K. Portacci, A. Reeves, B. Corso, and M. Salman, "Evaluation of vaccination strategies for an outbreak of pseudorabies virus in US commercial swine using the NAADSM," in *ISVEE 12: Proceedings of the 12th Symposium of the International Society for Veterinary Epidemiology and Economics, Durban, South Africa*, 2009, p. 78.

[5] Z. Sui, N. Harvey, and S. Pallickara, "On the distributed orchestration of stochastic discrete event simulations," *Concurrency and Computation: Practice and Experience*, 2013.

[6] M. Malensek, Z. Sui, N. Harvey, and S. Pallickara, "Autonomous, failure-resilient orchestration of distributed discrete event simulations," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, ser. CAC '13. New York, NY, USA: ACM, 2013, pp. 3:1–3:10. [Online]. Available: http://doi.acm.org/10.1145/2494621.2494625

[7] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[8] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[9] A. Bialecki, M. Cafarella, D. Cutting, and O. O'Malley, "Hadoop: a framework for running applications on large clusters built of commodity hardware," *Wiki at http://lucene. apache. org/hadoop*, 2005.

[10] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, pp. 59–72, 2007.

[11] S. Pallickara, J. Ekanayake, and G. Fox, "Granules: A lightweight, streaming runtime for cloud computing with support, for map-reduce," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. IEEE, 2009, pp. 1–10.

[12] D. Anderson, "BOINC: a system for public-resource computing and storage," in *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, Nov 2004, pp. 4–10.

[13] A. S. Tanenbaum, *Distributed operating systems*. Prentice Hall, 1995.

[14] M. Malensek, S. Pallickara, and S. Pallickara, "Exploiting geospatial and chronological characteristics in data streams to enable efficient storage and retrievals," *Future Generation Computer Systems*, 2012.

[15] ——, "Expressive query support for multidimensional data in distributed hash tables," in *Utility and Cloud Computing (UCC), 2012 Fifth IEEE International Conference on*, nov. 2012.

[16] R. M. Fujimoto, "Parallel discrete event simulation," *Commun. ACM*, vol. 33, no. 10, pp. 30–53, Oct. 1990. [Online]. Available: http://doi.acm.org/10.1145/84537.84545

[17] J. Misra, "Distributed discrete-event simulation," *ACM Comput. Surv.*, vol. 18, no. 1, pp. 39–65, Mar. 1986. [Online]. Available: http://doi.acm.org/10.1145/6462.6485

[18] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. Ieee, 2010, pp. 1–10.

[19] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal of Supercomputer Applications*, vol. 11, pp. 115–128, 1996.

[20] M. Litzkow, M. Livny, and M. Mutka, "Condor-a hunter of idle workstations," in *Distributed Computing Systems, 1988., 8th International Conference on*, Jun 1988, pp. 104–111.