Environmental Science

Volume 2, Issue 3, 464-493.
DOI: 10.3934/environsci.2015.3.464
Received date 28 January 2015,
Accepted date 18 May 2015,
Published date 8 June 2015

http://www.aimspress.com/

Research article

Encapsulating model complexity and landscape-scale analyses of state-and-transition simulation models: an application of ecoinformatics and juniper encroachment in sagebrush steppe ecosystems

Michael S. O'Donnell *

U.S. Geological Survey, Fort Collins Science Center, 2150 Centre Ave., Bldg. C., Fort Collins, CO 80526, USA

*** Correspondence:** E-mail: odonnellm@usgs.gov; Tel: +1-970-226-9407.

Abstract: State-and-transition simulation modeling relies on knowledge of vegetation composition and structure (states) that describe community conditions, mechanistic feedbacks such as fire that can affect vegetation establishment, and ecological processes that drive community conditions as well as the transitions between these states. However, as the need for modeling larger and more complex landscapes increase, a more advanced awareness of computing resources becomes essential. The objectives of this study include identifying challenges of executing state-and-transition simulation models, identifying common bottlenecks of computing resources, developing a workflow and software that enable parallel processing of Monte Carlo simulations, and identifying the advantages and disadvantages of different computing resources. To address these objectives, this study used the ApexRMS® SyncroSim software and embarrassingly parallel tasks of Monte Carlo simulations on a single multicore computer and on distributed computing systems. The results demonstrated that state-and-transition simulation models scale best in distributed computing environments, such as high-throughput and high-performance computing, because these environments disseminate the workloads across many compute nodes, thereby supporting analysis of larger landscapes, higher spatial resolution vegetation products, and more complex models. Using a case study and five different computing environments, the top result (high-throughput computing versus serial computations) indicated an approximate 96.6% decrease of computing time. With a single, multicore compute node (bottom result), the computing time indicated an 81.8% decrease relative to using serial computations. These results provide insight into the tradeoffs of using different computing

resources when research necessitates advanced integration of ecoinformatics incorporating large and complicated data inputs and models.

Keywords: STSM; high-throughput computing; high-performance computing; ecoinformatics; sagebrush steppe; juniper encroachment

1. Introduction

State-and-transition simulation models originated as conceptual representations of the different conditions (states) and changes in conditions (transitions) that characterize observations of natural ecosystems; modern data and computing capabilities have increased the quantification of these models. State-and-transition simulation models (STSM) require information about the initial conditions of vegetation to define states, transition paths between these states, and the behavior of these transitions, defined as deterministic and probabilistic pathways [1]. After researchers define their states, initial conditions, transitions, and pathways, they then use Monte Carlo simulations to measure the variability of the results that account for the stochastic properties of the model parameters. More complex models will include multiple scenarios that capture different management protocols [2], climate scenarios [3,4,5], state-and-transition attribute values (i.e., aggregating state classes and transitions to obtain different output summaries), various types of multipliers (e.g., the influence of slope on fire spread properties [6]), and stock-flow models (e.g., Intergovernmental Panel on Climate Change carbon models [7]). Although not required, STSM may also incorporate Geographic Information Systems (GIS) data of state classes, management zones, biophysical strata, and transitions.

Stochastic state-and-transition simulation models that are non-spatial or spatially explicit predict how the vegetation state classes change over time due to vegetation transitions resulting from natural and anthropogenic processes. These models include both processes (e.g., spread of fire and invasive species, encroachment resulting from attenuating natural ecosystem processes, and climate change [4]) and empirical observations (e.g., field observations indicating fire return intervals [8]). The resulting aspatial and spatial predictions describe changes such as habitat fragmentation, habitat quality, and vegetation composition and structure. State-and-transition simulation models are important because they produce information about natural and anthropogenic landscape changes. Researchers use scenarios to capture the effects of different practices for managing invasive species (e.g., cheatgrass and juniper invasion [9]), grazing practices, fire management [10], and land-use patterns [11]. These resulting predictions allow managers to explore effective management scenarios and the effects of alternative impacts to the landscape, assuming that the models capture realistic ecosystem processes.

The overall purpose of this research was to identify and develop different methods for running STSM with larger landscapes, higher spatial resolution data sets and models that are more complex. Specifically, the objectives of this work fit into two broad categories. The first category, understanding modeling and computing challenges, includes identifying (1) technical challenges of executing state-and-transition simulation models (2) common bottlenecks and limitations to effectively using computing resources, and (3) methods for addressing processing limitations. The second category, the integration of alternative computing environments to support state-and-transition simulation modeling, addresses (4) evaluating different computing resources that execute the decomposed Monte Carlo simulations, (5) developing software that decomposes the

state-and-transition simulation model as embarrassingly parallel Monte Carlo simulations, and (6) evaluating the advantages and disadvantages of running state-and-transition simulation models as embarrassingly parallel tasks. I discuss objectives 1-3 in the Background, objectives 4 and 5 in the Materials and Methods, as well as the Results, and lastly I discuss objective 6 in the Discussion.

2. Background

Until recently, scientists mostly relied on aspatial state-and-transition simulation models, and the software executed the model instructions in serial; that is, a single task executes before additional sequential tasks run, but none executes concurrently and software cannot take advantage of multiple processor cores. Representing spatial data in STSMs is necessary for capturing spatial processes [12] and multi-scaled processes [13]. In this context, scale refers to the combined effect of spatial extent and resolution (grain), while when referring to the scaling of computers and computer infrastructure, scaling denotes the expansion of computing resources. Research requiring complex models and regional GIS data could encounter unrealistic computation time while using serial computations, which usually results in curtailing hypotheses. Specifically, the researcher might change the number of states represented in their model, the types of transitions, the number of Monte Carlo simulations, and other factors used to define a model. In turn, a researcher may be forced to change the questions they investigate due to the computing limitations imposed on their modeling procedures.

To address this challenge, this study explored decomposing Monte Carlo simulations of STSMs into parallel tasks. ST-Sim™, a plugin for ApexRMS SyncroSim™ software (freeware [14]), supports decomposition of models on a single multicore machine. This study demonstrated that when a vertically-scaled computer, such as a single desktop, cannot sufficiently handle complex models while using parallel tasks, scientists can rely on distributed computing environments, also known as horizontal scaling. High-throughput (HTC) and high-performance (HPC) computing are methods of distributed computing that enable researchers, modelers and analysts to distribute their workload on multiple machines, thereby supporting analysis of larger landscapes (spatial extents), higher spatial resolution data inputs, more complex models, and sufficient Monte Carlo simulations in a timely manner.

2.1. Challenges of processing state-and-transition simulation models

State-and-transition simulation models include a myriad of parameters that may present as challenges when executing these models with standard computing resources. For example, models often require many Monte Carlo simulations to quantify effects of parameter variability on simulation results. They may also include many time-steps, which must reflect an appropriate length of time specific to the plant community life cycles. Model scenarios behave independently of each other, and researchers evaluate the juxtaposed outcomes to understand relations between parameters and community responses. Thus, with additional scenarios, each with a set of Monte Carlo simulations and time-steps, the computation time and model complexity increases. When using spatial models, large extents and small spatial resolution (grain) can contribute to increased computing time. State-and-transition simulation models also support spatial autocorrelation of vegetative transitions, such as spreading of fires [15] because with these parameters, models can more accurately capture how the landscape is likely to change through time. However, with the increased complexity to represent these processes accurately, the greater the computations and the need for different computing resources. Thus, expanded ability to represent complex ecological

scenarios comes with computational costs that may challenge practicality and efficiency.

Scientists often address computing bottlenecks by decreasing the model complexity. With spatial STSMs, one can reduce the spatial extent and/or spatial resolution (e.g., large grain) , reduce the number of Monte Carlo simulations, test fewer scenarios, reduce the number of strata, and change the research questions. However, the scientific community can often address limitations of computational runtime because of the abundantly available and inexpensive computing hardware, as well as the many different methods available to solve problems (e.g., in-memory computing, many task computing, HTC, HPC, and big data applications). Frequently, computer scientists use parallel computing, as described here, to address large problems by dividing these into smaller pieces and solving them concurrently. Decomposition of a problem can take advantage of multiple processors by breaking down the data into smaller pieces (*data decomposition*) or by using an algorithm (*functional decomposition*) that divides the computations into smaller pieces [16]. Here, I introduce these concepts and explain where scientists can access additional computing resources and how they can use them without having to change the research.

Several potential methods exist for decomposing state-and-transition simulation models. The first method, and the most obvious method, is to run each Monte Carlo simulation independently (data decomposition). This approach does not require complex programming because no communications between the Monte Carlo simulations are necessary. A second approach to decomposing the problem includes dividing the spatial data into tiles and then running the STSM on the individual tiles (data decomposition). This approach is difficult because in many cases, these models will include spatial relationships between neighboring pixels that would result in errors along adjacent tiles. A third approach includes decomposing the state-and-transition algorithms (functional decomposition). For example, a common approach of functional decomposition is using the message-passing interface (MPI) to loop through arrays of data (such as those in a raster data set). The data is decomposed into arrays using a loop function, then this data is processed on remote central processing units using some algorithm, and finally these results are merged into a single raster data set. The decomposition and aggregation of the data relies on a significant amount of communication streaming between the different processors and compute nodes [17] to accomplish the processing, which sets this approach apart from embarrassingly parallel methods that use no communication between computing tasks. *Embarrassingly parallel* is a term used to denote programming tasks or processes that require solving many similar but independent tasks simultaneously, with little or no coordination needed between the tasks. The message passing interface approach can also be difficult to achieve because of the prevalent dependencies between neighboring pixels. The goal here was to demonstrate the decomposition of Monte Carlo simulations for STSMs and execute these as embarrassingly parallel tasks. For more complex models (not demonstrated here), this workflow may be applied to individual scenarios and strata. For example, like Monte Carlo simulations, the scenarios are independent of each other, and each scenario as well as its associated Monte Carlo simulations can run in parallel. Because there is no underlying difference in parallelizing the scenarios and the strata, this was not explicitly investigated.

2.2. Identifying computing bottlenecks

Numerous computer hardware components can potentially affect model performance. These components include Computer Processing Unit (CPU) speeds, the number of processor cores, system

bus speeds (communication between CPU and memory), and storage configuration. Beyond a computer, networked storage, the bandwidth, and the configuration of networks can influence how long a model runs [18]. In addition to potential hardware and network bottlenecks, workflows and the algorithms used within software can also lead to poor performance. Isolating bottlenecks and poor performance is not an easy task, especially if you are not a computer scientist, a software engineer, and you do not have access to the software code.

Understanding computer-hardware configuration helps identify software capabilities and requirements. For example, a *multi-CPU* (symmetric multiprocessing system) will have multiple and identical processors located on different circuits, all sharing memory via a *system bus*; however, each CPU has its own memory cache [18]. The system bus connects a computer's motherboard to the CPU and memory. A *core* is a computational unit of a multicore processor (single CPU), and a *multicore chip* has a single socket that combines two or more independent processors on the same circuit and with a memory cache [18]. For example, these components may be configured to create dual core processors (2 cores per physical processor) or quad core processors (4 cores per physical processor), which are both multicore chips. The performance enhancement of multicore chips is significantly less than adding an additional CPU chip [19]; however, the benefit is less electrical power consumption and reduced overhead for bus communication [20]. Hyper-threading is a Microsoft Windows® BIOS (Basic Input-Output System) configuration that allows one physical processor with multiple cores to appear as multiple logical processors. The more CPUs, cores, and memory a machine has, usually the faster the processing. The processor speed and type are also important considerations as they both can affect processing performance. Configuring a single system and providing adequate memory is important when scaling hardware, but software must have the ability to use these resources.

When the network infrastructure and computer hardware are not the bottlenecks, or when the cost of reducing these bottlenecks is impractical, one can consider the software architecture (32-bit versus 64-bit) as a performance inhibitor. SyncroSim supports both 32-bit and 64-bit architectures and the architecture is important because it defines how much data the software can load into memory at once. The more data stored in direct access memory, the better the software performance [18] because it reduces the number of reads and writes to disk and the number of times information is transferred across the system bus. The Direct (main) Random Access Memory (DRAM, level 3) of a computer can support up to 2^{32} bytes for 32-bit architectures and 2^{64} bytes for 64-bit architectures. A 32-bit architecture allows for up to 4 GB of RAM and a 2 GB limit per process. A 64-bit architecture allows for up to 256 TB of RAM and an 8 GB limit per process. Thirty-two bit applications work on 64-bit platforms because there is an emulation layer but the same 2 GB limit per 32-bit application applies. If a model requires processing a significant amount of information by loading more than 2 GB of data into memory, then a 64-bit architecture is necessary. Numerous facets to architectures and software performance exist, and limitations exist for operating systems with respect to how much one can scale (i.e., expand resources) memory and CPUs [21,22]. Monitoring memory usage relative to memory limits can help isolate this bottleneck.

Understanding the basic concepts of computer architectures will help researchers to begin understanding the various hardware components and complexities. Knowing your hardware configurations provides insight into how a researcher can address larger computing problems. Here, I have defined cores, multicores, multi-CPUs and memory architectures, which are required to understand the different methods for addressing processing limitations. Accepting the abilities and

limitations of the SyncroSim software, I explored effects of decomposing the model in different computing environments to address potential computing bottlenecks that may occur because there are too few resources for addressing the problem.

2.3. Methods for addressing processing limitations

Parallel computing addresses computational challenges by concurrently executing tasks (e.g., a program that includes many instructions) using two or more computer processors and/or nodes (i.e., computers). A variety of parallel computing methods exists, and these include bit-level, instruction level, thread level, data, and task parallelism. These are concepts not usually known to ecologists, and therefore, I have provided a brief summary as several pertain to this research. The difficulty of implementing parallel computing is addressing the communication and synchronization between the subtasks, which requires a knowledge of the methods for decomposing the instructions and/or data and deciding how these interact. Parallelization is categorized into fine-grained parallelism, which supports communication many times per second and coarse-grained parallelism, which requires little or no communication between concurrently running subtasks.

Bit-level parallelism refers to how a computer processor executes instructions (for example, an algorithm) as well as how the number of instructions and the size of those instructions affect this execution. If a computer processor requires splitting a request into sequential instructions because the instructions do not fit into a single bit array (e.g., 32-bit) then processing time increases. If the instruction fits into a larger bit array (e.g., 64-bit) then processing time decreases. Instruction level describes how a software developer manipulates the order and grouping of a program's instructions. Each stage of a multi-stage instruction pipeline represents what the processor does to the instruction. For example, these stages might include fetching the instruction, decoding it, executing it, and then accessing the memory. Superscalar processors (specialized processors), for example, execute multiple pipeline instructions concurrently. Threads are a simpler method for parallelization because they have their own memory stack and instructions, but each thread must wait on another thread because they are writing to the same memory address. Threads are commonly used with a graphical interface where a user executes a process, and the software then displays a progress bar to monitor the state of the process. In this example there are two concurrent threads running (e.g., a task/job and a progress bar). Threads are not used for most parallel computing applications because the threads belong to a single process, which does not support CPU intensive applications. Data parallelism uses a single calculation on different subsets of data, which is commonly used with loops within software. Task (i.e., a collection of instructions such as a software program) parallelization performs different calculations on the same data or on different data.

In this study, I am using bit-level (64-bit software and computer architecture), thread-level (streaming standard output for handling asynchronous logging of tasks), and task parallelism (parallelization of the STSM Monte Carlo simulations). The decomposition of the Monte Carlo simulations used in STSM requires no data dependencies or communications between the executions of tasks. The computer science community refers to this type of parallelization as embarrassingly parallel. Executing Monte Carlo simulations in parallel, which I have applied in this research, is a common example of embarrassingly parallel applications. Contrary to coarse-grained parallelism, high performance computing relies on a significant amount of communication and dependencies between subtasks, which this research does not implement.

To understand the limitations of parallel computing with respect to computer hardware, Flynn's taxonomy [23,24] has been used to categorize computer architectures into four categories. Since Flynn's work, two additional subcategories have been identified. All categories refer to the number of concurrent instructions that can process on the available data streams. Understanding Flynn's taxonomy is important with respect to parallel computing because parallelizing tasks are constrained by the computer's architecture and modeling environment. Flynn's taxonomy defines Single Instruction and Single Data (SISD), Single Instructions and Multiple Data (SIMD), Multiple Instructions and Single Data (MISD), and Multiple Instructions and Multiple Data (MIMD). Two additional categories that the informatics community later recognized include the Single Program and Multiple Data (SPMD) [25] and Multiple Programs and Multiple Data (MPMD) [26]. Software developers and researchers need to understand their hardware, parallel computing approaches, and their model structure before making informed decisions on improving modeling performance.

Of the six taxonomy classes, most are not applicable to how this study decomposed the STSMs, or applicable at all. SIMD are common with General Purpose Graphic Processing Units (GPGPUs) that impose a lockstep where each CPU instruction processes multiple data elements at the same time. MISD are used for high fault tolerance applications via redundancy of instructions (a common example for this type of hardware and computing are flight control computers). An average desktop workstation is indicative of the SISD category and for all purposes reflects the framework used for the STSMs. Networked clusters, grids, multi-processor computers, and multicore computers that use MPI [27] fit the MIMD taxonomy, but they also support embarrassingly parallel applications. *Grid computing* refers to processing jobs across multiple distributed computing systems that use different management configurations. That is, while multiple collections of computers may be managed autonomously, these collections of systems can also operate collectively.

Embarrassingly parallel workloads require minimal effort for decomposing a problem into parallel tasks and have little to no dependencies between tasks, such as communication. Consequently, these processes are common for grid computing because this type of computing relies on little or no communication and no dependencies exist between tasks. Embarrassingly parallel is similar to SPMD, a modification of SISD [28]. Embarrassingly parallel problems scale linearly if all resources are equal (e.g., processor speeds, memory caching and availability, bus connections, disk input/output speeds), while functional decomposition does not generally result in linear scaling. This is because functional decomposition uses MPI to process data and communicate between finer scales that result in non-linear scaling [29]. Although a cluster can support MPI, I used the cluster to execute embarrassingly parallel tasks that require no communication, which means the cluster cannot perform the tasks faster than grid computing if all hardware is identical. Since this work uses embarrassingly parallel computations, I expected the results to scale linearly.

An Information Technology (IT) administrator can scale computing resources through vertical scaling and horizontal scaling. *Vertical scaling* (scale-up) refers to adding resources to a single node (i.e., workstation or a machine within a cluster [28]). For example, one can expand a computer's processing capabilities by adding CPUs, replacing a single core CPU with multiple core CPUs, altering the CPU configuration (e.g., hyper-threading), adding memory, and using specialized hardware (e.g., general purpose graphical processing units [30]). *Horizontal scaling* (scale-out) refers to adding additional nodes to a networked system, such as, adding machines to an Local Area Network (LAN), Wide Area Network (WAN), or cluster [28]. Tightly coupled multiprocessor systems have multiple CPUs connected to the system bus and loosely coupled multiprocessor

systems have multiple computers with multiple CPUs interconnected via the Ethernet. The former is used for frequent communication between processes (e.g., High-Performance Computing [HPC]), while the latter are used when little or no communication is necessary (e.g., High-Throughput Computing [HTC]) or when significant levels of communication are necessary [28]. HPC also supports large amounts of memory and CPUs as shared resources, while HTC takes advantage of many CPUs without the ability to share resources [28]. In other words, HTC systems appear as individual networked machines and HPC systems appear as a very large single machine when the software interacts with the hardware. Many different methods exist for configuring both HTC and HPC systems and therefore, these are gross simplifications to characterize the main differences.

High-throughput computing, also known as distributed computing and grid computing, uses a collection of low-cost commodity computers to increase capabilities of large-scale computational problems with multiple independent job instances. This is because HTC consumes a set of independent jobs with no shared states and processes these jobs concurrently on remote machines (within a LAN or across the WAN). Although numerous configurations of grid infrastructures exist, generally each hyper-threaded core or CPU on a single compute node behaves as a computing slot where a single job executes independent of other jobs. Because grid computing is not constrained to homogeneous infrastructures, management configurations, architectures or geographic location, they can support a more diverse set of applications but at the cost of being more difficult to manage and operate. An HTC system requires middleware (open source or commercial software), which is a job management resource installed on each node. For example, the University of Wisconsin developed HTCondor™ middleware (open source [31]), which has a long-standing history and a broad use in the university and research communities. Other examples of open source and commercial HTC systems include Unicore™, Globus™ Toolkit, Univa®, NorduGrid™, and Appistry®. With these management systems, a user submits jobs from the submit node and jobs run on the compute nodes. Computers can also have a combination of roles and a single machine within the system, the central manager, manages all clients, jobs, matching of jobs to clients, job scheduling, and related intricacies. Many configurations of these systems exist, but these are the basic principles.

High-performance computing, like high-throughput computing, requires middleware for job and resource management. Common software used for job scheduling of HPC systems are the Portable Batch System (PBS), Torque resource manager, Simple Linux Utility for Resource Management (SLURM) and OpenPBS. The installation, configuration, operation, and functionality of the middleware are similar to HTC middleware. Like HTCondor, a submit file is created for executing a job. With the submit file, the user specifies the required number of nodes, processors, and library requirements. When a researcher uses HTC, they must define all the parameters that their software requires, and they must define how, when, and where the data and software will reside and interact with the remote computers. Remember, HTC computing resources do not require that the systems exist on the same LAN, much less the same geographic location (i.e., accessible via the WAN), but transferring data across the WAN is likely a bottleneck for STSMs relying on significant data inputs and outputs. The primary difference between HTC and HPC with respect to submitting jobs is that HPC management systems do not require information pertaining to heterogeneous infrastructures (e.g., hardware and operating system differences).

High-performance computing systems are generally more expensive to purchase compared to HTC. Modeling on HPC systems mostly focus on finer scale parallel computing, such as functional decomposition, while HTC focuses on throughput (volume). With embarrassingly parallel tasks, the

model framework is high-throughput. HPC infrastructures support parallel tasks that exchange data during the computation, while loosely coupled systems support executing parallel tasks independent of each other. Users typically wait in a queue (similar to a print queue) longer on HPC systems before their jobs execute because there are fewer HPC systems available due to costs. The primary difference between HTC and HPC is that HPC lends itself to applications requiring greater communication between processes, greater memory per instruction, shared states, and software specifically written and compiled for parallelizing the data and instructions. Therefore, users will benefit from understanding what computing environment they require, keeping the costs in mind for using and operating different computing environments. For this study, I used a single machine with multiprocessors and multicores as one environment, a high-performance computing environment (cluster), and a high-throughput computing environment to execute the STSM Monte Carlo simulations as embarrassingly parallel tasks. These environments support embarrassingly parallel computations of the STSMs. Embarrassingly parallel computations were an obvious approach to decompose the STSM, which did not require a re-write of the STSM software.

3. Materials and Methods

3.1. Study area and state-and-transition simulation model case study

The study area was located in southwest Idaho (Castle Creek), USA in a semi-arid shrub-steppe ecosystem (Figure 1). The vegetative strata for the ApexRMS case study include "Basin Big Sagebrush Upland" (*Artemisia tridentata* ssp. *tridentata*), "Curleaf Mountain Mahogany" (*Cercocarpus ledifolius*), "Low Sagebrush" (*Artemisia arbuscula* Nutt.), "Montane Sagebrush Upland" (*A. tridentata* ssp. *vaseyana*), " Montane Sagebrush Upland With Trees", "Western Juniper Woodland & Savannah" (Western Juniper: *Juniperus occidentalis*), and "Wyoming and Basin Big Sagebrush Upland" (Wyoming Big Sagebrush: *Artemisia tridentata wyomingensis*). ApexRMS provided the Castle Creek case study (ST-Sim-SpatialSample-V2-1-0 [32]), which included the data sets and STSMs. I evaluated the "Spatial – Current Management" scenario on the Castle Creek landscape where management treatments included thinning encroaching western juniper. Historically in the Castle Creek landscape, juniper occurred in rocky areas and cooler areas where fires did not occur [33]. The distribution of juniper increased because of fire suppression, grazing, and climate change [34]. With increasing juniper, habitat quality for wildlife, such as Greater Sage-grouse (*Centrocercus urophasianus*), and grazing resources for livestock have decreased [35-38]. I used these data and models to evaluate the decomposition of STSMs in different computing environments.

The state-and-transition simulation model for the case study includes deterministic and probabilistic transition pathways (Figure 2). Transitions are necessary to define the biotic and abiotic events that result in observable community changes, which occur from natural and anthropogenic (e.g., management) influences. Deterministic transitions define the transition paths between class states that occur at different succession stages [10], which are documented and understood.

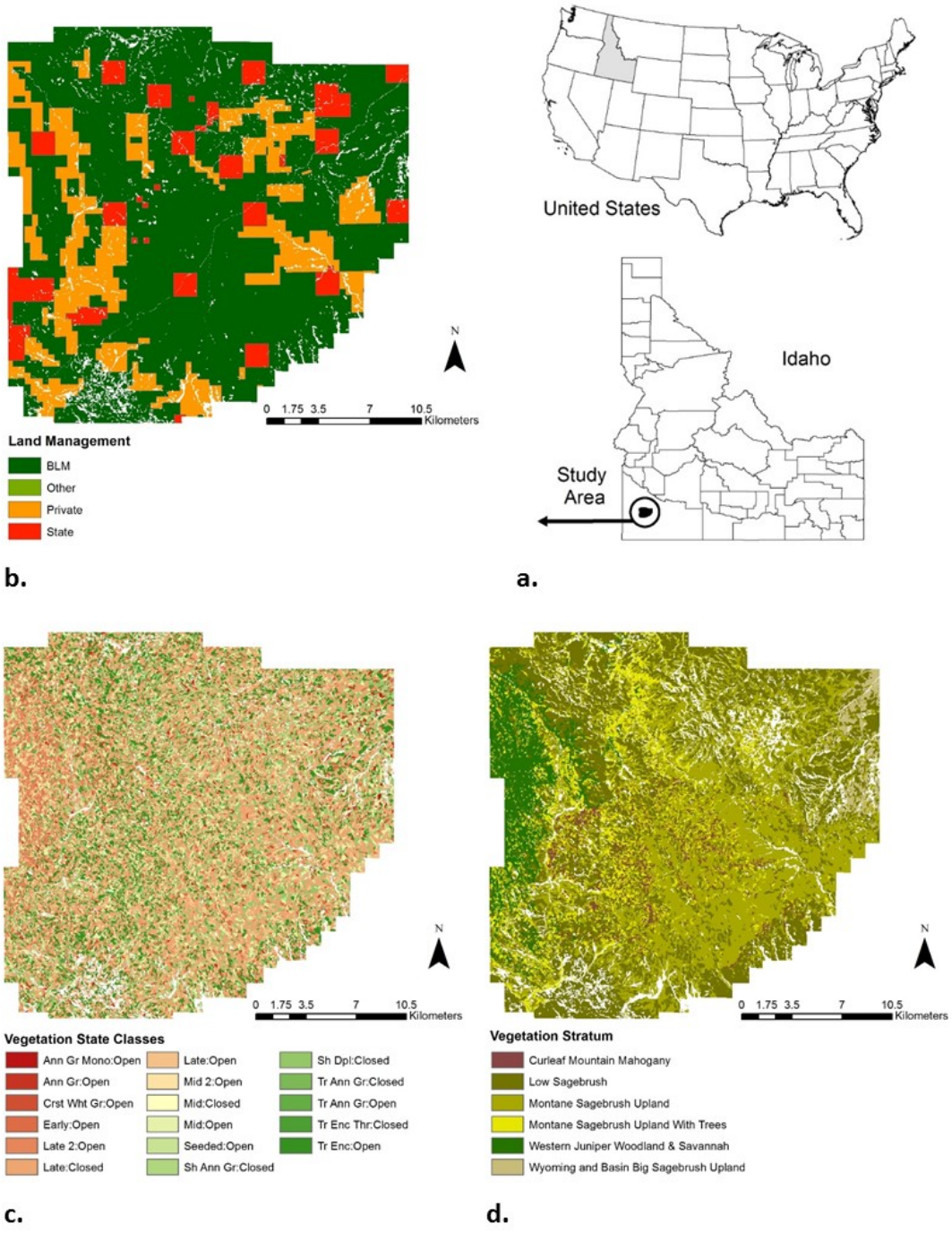

Figure 1. (a) Locational diagram of the study area in Idaho, USA; (b) Land management delineation for the Bureau of Land Management (BLM), private lands, state lands, and other. Most removal of juniper encroachment will occur on public land, but there are also conservation plans that provide economic incentives to improve habitat; (c) The vegetation states define the predominant vegetation type and structure (i.e., age). A few definitions of these referenced vegetation classes found in the Geographic Information System (GIS) data include annual grass (Ann Gr), tree (Tr), encroached conifer (Enc), mountain big sagebrush, native herbaceous cover, conifer cover (ShDpl), greater 50% native herbaceous cover and less than 10% cover mountain big sage (Early open); and (d) Within the state-and-transition simulation models, the vegetation strata represent the spatial aggregation where vegetation states change through deterministic and probabilistic pathways.

The probabilistic transitions capture changes between vegetation classes, which rely on empirical studies [39] that identify the uncertainty of changes between states based on particular influences not well understood but observed or modeled in studies. Transition multipliers (i.e., transition frequencies through time [6]) and directional multipliers (i.e., influence transition spread rates) are additional, ecologically useful methods, which also increase model complexity; I do not discuss these applications in detail here, because they do not uniquely influence the methods for decomposing the STSMs for parallel computations. State-and-transition simulation modeling is complex and researchers are required to generalize what they understand about transitions and model parameters. The models are generalizations of ecosystem processes, and this simplification allows researchers to model and evaluate those models, with observations and by testing different parameters and scenarios. These data and models support the relative complexities of STSM design and they provide a realistic case study to explore the benefits of decomposing STSMs and using alternative computing environments.

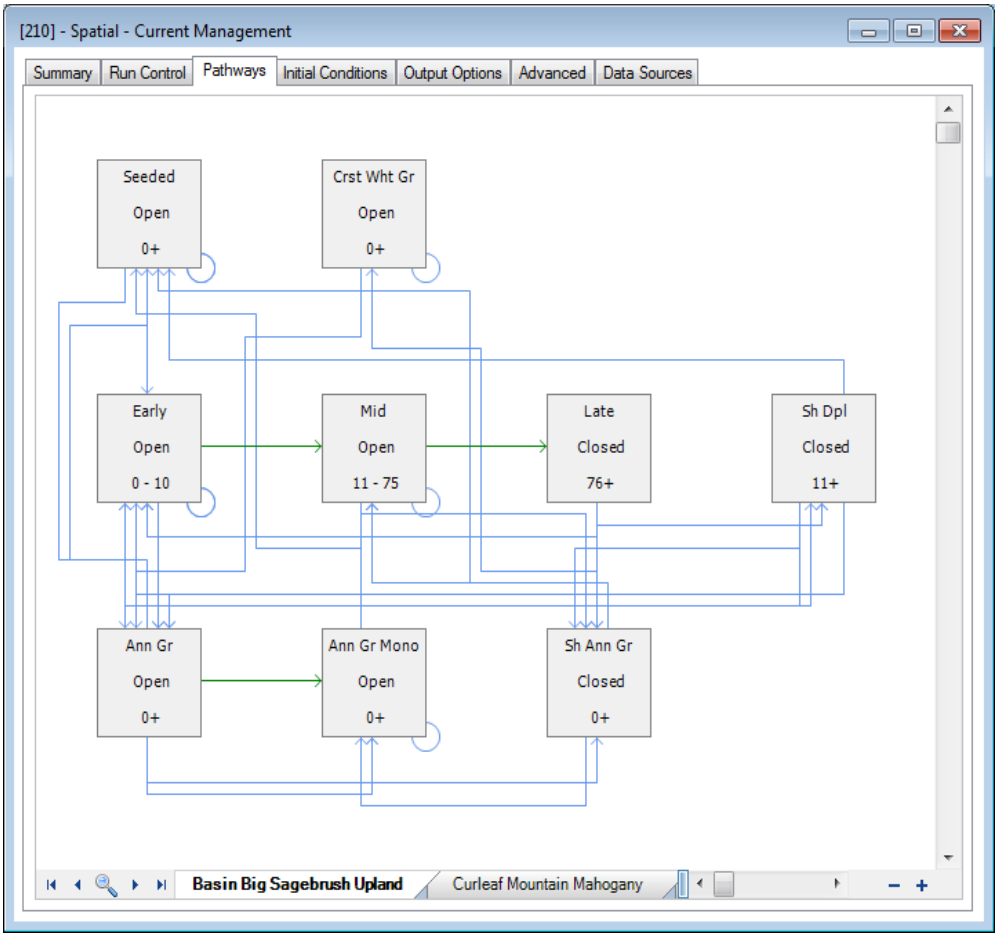

Figure 2. The basin big sagebrush upland stratum as defined in the Castle Creek study and this illustrates the pathways between its state classes. The green lines represent deterministic transitions and the blue lines represent probabilistic transitions.

3.2. Software tools and computing environments

The native software environment of SyncroSim ST-Sim is .NET (Microsoft® NET Framework). The researcher must use the ST-Sim Graphical User Interface (GUI) to parameterize STSMs, but recent releases of the software include a command line interface for executing, but not parameterizing, simulations. The command line software also operates on Linux when executed from Mono™ [40]. Mono is an open source project that enables Microsoft .NET applications (specifically C Sharp) to operate on cross-platforms, such as Linux distributions. Since most grid computing and high-performance computing environments rely on Linux operating systems, researchers can take advantage of these resources with the command line version of ST-Sim. Because I have compared the performance of ST-Sim on a multiprocessor machine to the performance within an HTC and HPC environment, having the ability to execute tasks on Linux operating systems was extremely important.

I used two resource managers (i.e., middleware that allow users of HTC and HPC systems to interact with the networked systems) for deploying the embarrassingly parallel computing tasks on HTC and HPC environments. HTCondor is an open source grid computing resource manager, and the Simple Linux Utility for Resource Management (SLURM [41]) is an open source resource manager for Linux HPC clusters. I used PuTTY [42] as a terminal emulator to log onto the remote HPC Linux cluster via a secure shell (SSH) protocol, and I used WinSCP [43] to transfer the software and data from a local Networked Attached Storage (NAS) device to a local NAS on the Linux HPC cluster. I used the ST-Sim-V2-3-5-x64 software version to analyze the STSMs, which supports a 64-bit architecture.

3.3. Multicore desktop computing

I executed and then compared the runtime of the Castle Creek STSM using five different computing environments (Table 1). The first environment used serial computations via the SyncroSim ST-Sim GUI. With the first environment, I ran a simulation for a single replication to establish a runtime for each replication. I also compared this runtime to a single replication using Mono on a Linux virtual machine to assess whether Mono influences the performance. I would expect all parallel computations to complete within a timeline equal to the baseline runtime times the number of simulations. Most software that uses data read/write operations would perform best when the data is closest to the CPU. Executing a model with locally stored data also eliminates network bottlenecks as a culprit. Therefore, one can establish a baseline of performance by running the software with the data stored on a local machine and then compare its runtime when storing the data on a networked storage device. Because most networked storage uses Redundant Array of Independent Disks (RAID), hybrid storage (hard disk drives and solid state drives), or high performance storage (parallel read and writes), performance may be worse, and sometimes, storing data locally is not possible because most desktop machines have limited storage capabilities. Regardless, this test can assist with removing factors related to the network and storage. Because the resulting data for the Castle Creek study area required approximately 146 GB of storage, using local storage to test network related bottlenecks was not possible with the available hardware configurations.

The second environment also used serial computations via the GUI, but instead of running a

single Monte Carlo simulation, I used 50. This baseline establishes a runtime to compare to all parallel computing tests. The third environment executes 50 Monte Carlo simulations on a single local machine with parallel computing. Here the number of simulations is strategically greater than the number of available cores. The fourth environment was an HTC system, and the fifth environment was HPC, both environments used parallel computing and 50 Monte Carlo simulations of the Castle Creek study area. I had no control over which nodes within the cluster executed the Monte Carlo simulations, but I distributed the jobs across the nodes evenly and during times of inactivity so I was not competing for resources. As for HTC, I also submitted all jobs during a time that processing did not compete for computing resources. Importantly, and as mentioned before, all computations used embarrassingly parallel computations and not MPI computations.

Table 1. Distinction of the simulations compared in this study using the different combinations of replication, hardware, and software.

Title	Replication Count	Description	Hardware	Available Node Count	Available Core Count	Available Memory	ST-Sim Software	Resource Manager
GUI Baseline	1	a	Single desktop[1]	1	1	26 GB	ST-Sim GUI	
Serial	50	b	Single desktop[1]	1	1	26 GB	ST-Sim GUI	
Parallel (Local)	50	c	Single desktop[1]	1	16	26 GB	ST-Sim command line	
HTC	50	c	Multiple desktops HTC (LAN)[2]	50	395	2 GB/ core	ST-Sim command line	HTCondor
HPC	50	c	Cluster[3]	54	609	2.3 TB	ST-Sim command line	SLURM

1. Single desktop simulation used had two available logical processing units (3.0 GHz);
2. The Multiple desktop simulations HTC (LAN, local area network) used similar hardware to the Single desktop environment, but many desktops available to handle the distributed jobs. This HTC system included a local NAS with 5 TB and 1 GB/s Ethernet interconnect.
3. The Cluster simulations used a local NAS with 45 TB of storage and 1 GB/s Ethernet interconnect while the processor speeds varied between 2.7-3.0 GHz.

a. A single Monte Carlo simulation.

b. Multiple Monte Carlo simulations (executed in serial).

c. Multiple Monte Carlo simulations (executed in parallel).

3.4. Decomposition workflow

The workflow for decomposing and concurrently running the Monte Carlo simulations follows a simple structure (Figure 3). First, I set up the STSM scenario, and then I tested this model by

running a single replication. I then investigated these results and determined whether the results appeared realistic. After I established a working model, I started with scaling the workflow, which I accomplished by using three basic steps (Figure 3). The decomposition (Step 1 of Figure 3) refers to splitting the STSM into independent components and treating each Monte Carlo simulation and the data they require as independent models. For example, the decomposition, which is handled by the ST-Sim software, creates a copy of each STSM database and all GIS data. Step two (Figure 3), launches the Monte Carlo simulations as independent processes. The last step of this workflow (Step 3 of Figure 3) includes merging all the returned results from each Monte Carlo simulation so the final ST-Sim database contains all results. The output GIS data sets depend on the STSM configuration settings. For example, the researcher may want aspatial and spatial summary results for every time-step of all state classes, transitions, state attributes, and transition attributes. For these tests, I returned aspatial and GIS results every five time-steps. Each test used a spatial model with 150 time-steps, the data input required 0.3 GB of storage, and the output required 146 GB.

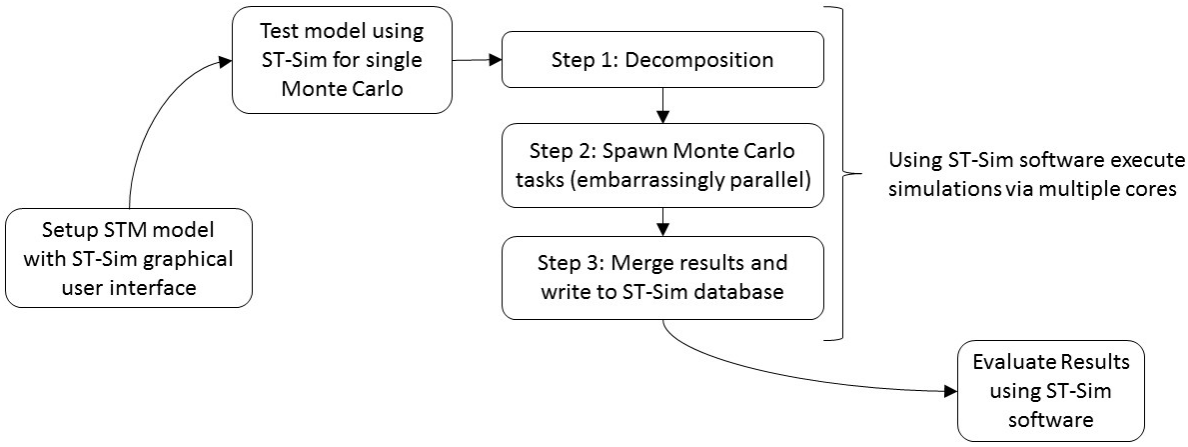

Figure 3. The parallel computing workflow for a multicore desktop sets the stage for how the researcher will accomplish these tasks in grid computing and high-performance computing environments.

3.5. High-throughput computing

High-throughput and high-performance computing require more elaborate workflows than using a single multicore machine because they use resource managers that help users interact with a complex system. I used the basic workflow of figure 3 and then built on these concepts to develop the HTC/HPC workflows. To test the Idaho case study in an HTC environment, I developed a workflow (Figure 4), two Python® scripts, and an HTCondor template submit file. First, I developed an HTCondor template submit file that provides the submit file parameters. Within this file, I included configuration settings defining the runtime environment, the streaming of logging, operating system and hardware requirements, rules to transfer software and data, email notification alerts, and content that is specific to the job running on the client node. The purpose of the submit file template is to allow researchers to define parameters that change between HTC systems due to different management protocols defined by an IT administrator without having to modify the Python

Script 1. The user should modify the parameters within the template based on their system's configuration. Many configurations exist for HTC systems, and the HTCondor template supports this flexibility. After configuring the submit file template, the researcher executes the Python Script 1 that sets up embarrassingly parallel computations for grid computing (Step 2 of Figure 4). During this step, the decompositions of the STSM and data are processed and the script generates HTCondor submit files. The researcher may also decide to zip and transfer their STSM data to the client or not transfer the data and instead use centralized storage. Transferring of data is a parameter within the Python Script 1 that the user can change.

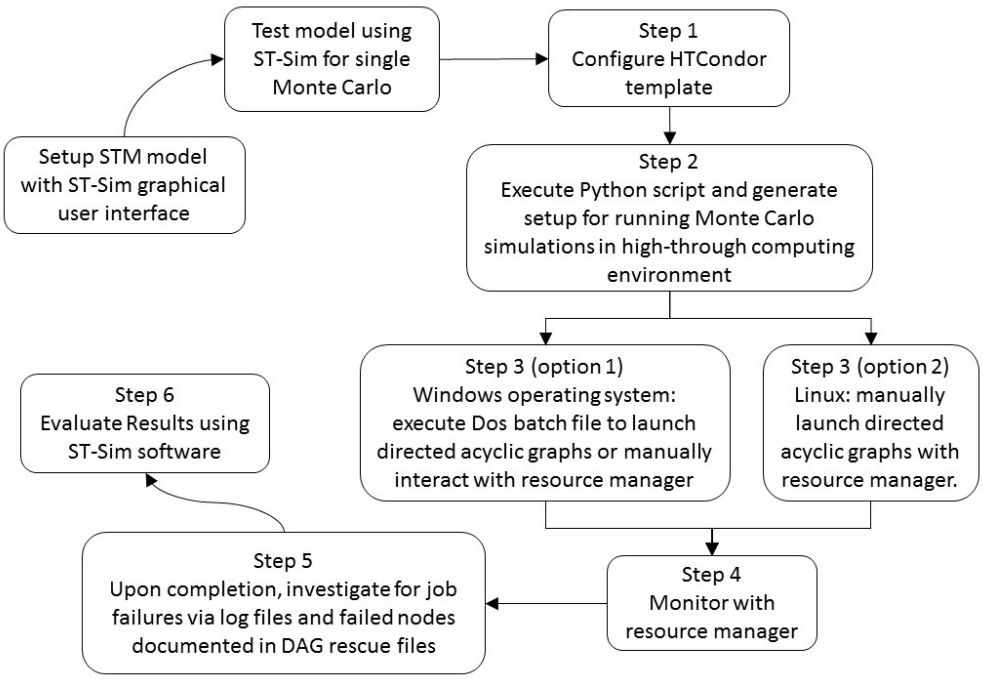

Figure 4. The workflow for executing state-and-transition simulation models in a high-throughput computing environment while using HTCondor middleware is the more complex system to use, but its complexity allows the middleware to take advantage of heterogeneous computing environments.

The Python Script 1 generates a Directed Acyclic Graph (DAG) file, which defines how the individual submit files are executed and any relationships between the tasks. DAGs are a data structure (Figure 5) where each job represents a node and each node is a parent or child. The child nodes are dependent on parent nodes. If no dependencies exist, then you have only parents. I used DAGs to help reduce the overhead of detecting failed jobs as well as to increase flexibility of grid computing configurations. After I executed the Python Script 1, I submitted the HTCondor DAG (Step 3, Figure 4). This is possible using a Microsoft Windows® DOS batch file (generated from Python Script 1) or via command line on Windows and Linux operating systems. If Linux is used, then Mono is required on each client of the grid system and modifications to the Python Script 2 are necessary (not currently supported, but the modification is minor and similar to the syntax used in the HPC script). While the jobs run, progress is monitored (Step 4, Figure 4) using HTCondor or other third-party software. The Python Script 2, which runs on the client, is compiled so a Python

install is unnecessary on each client. This script handles the processing of Monte Carlo simulation tasks and streams the standard output and error to log files using multithreading, a different form of parallel processing. Step 5 (Figure 4) requires the user to investigate for job failures using log files that the resource manager documents within the DAG log and rescue files. Failed jobs are easily addressed by manually re-submitting the original DAG, which is only required when a DAG rescue file is generated. The final step (Step 6, Figure 4) requires the researcher to use the ST-Sim GUI software and evaluate the modeled results.

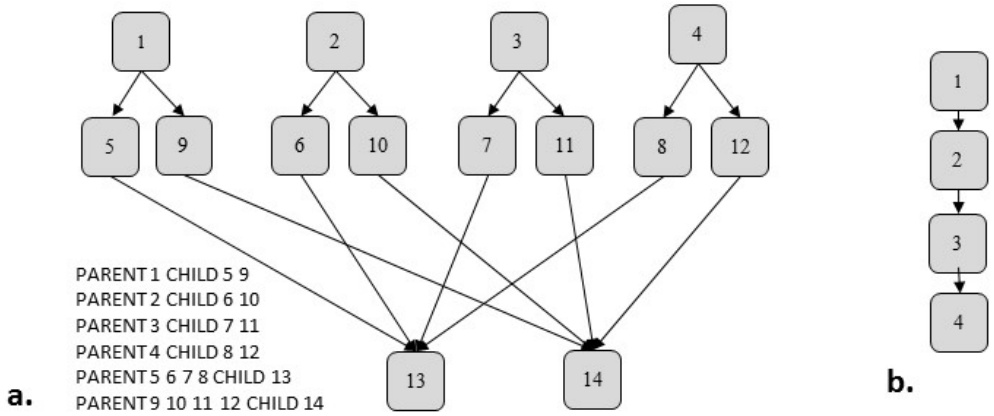

Figure 5. (a) This is an example of a more complex Directed Acyclic Graph (DAG) shcematic. The numbers represent a unique identifer for each job within the DAG. The links represent the parent-child relationships (i.e., the relationships between tasks and therefore a workflow for when jobs execute). **(b) This DAG illustrates what is used for the state-and-transition simulation model.** With this approach there are no children and therefore the DAG is not necessary, but it is useful for managing the jobs. The text on the lower left portion of (a) highlights the content within a HTCondor DAG and how the relationships are defined. This information is not found in (b) because there are only parents.

3.6. High-performance computing

The workflow for using HPC and submitting the embarrassingly parallel state-and-transition Monte Carlo simulations is significantly simpler than HTC because the system is homogeneous. Instead of using Python scripts and submit files, the approaches used for HTC, I manually wrote a text file that uses SLURM. This is a straightforward script to develop and it includes the three basic steps that were used across each computing environment (Figure 3). Because a cluster can use different resource managers (e.g., SLURM and PBS) and configurations, this template is a guideline that a researcher can use with different resource managers. For example, the cluster used in this study did not support SLURM arrays, so instead this script used a for loop to spawn each Monte Carlo simulation. Users can modify this template as well as convert its workflow to use other resource managers.

The cluster used for this study did not provide Mono as a software resource, so an IT administrator installed this as a module that SLURM can load during the execution of each job. After developing the SLURM script, I tested the command line version of SyncroSim ST-Sim on a linux

virtual machine located in-house. Here, I was able to verify that Mono was correctly compiled and that the ST-Sim command line could run under Mono. I then transferred the SLURM script, ST-Sim command line software, and the Castle Creek STSM data (database and GIS data) to the cluster storage using WinSCP as illustrated in the workflow developed for this study (Step 1 and 2, Figure 6). Using PuTTY, I remotely logged on to a terminal and executed the SLURM script with the sbatch command (Step 3, Figure 6). I monitored the progress of the simulation runs and then manually reviewed the log files to determine if any tasks failed (Step 4, Figure 6). The resulting data were transferred back to the local work environment using WinSCP (Step 5, Figure 6). Once the data resided on a Microsoft Windows environment, I evaluated the results in the SyncroSim ST-Sim GUI (Step 6, Figure 6).

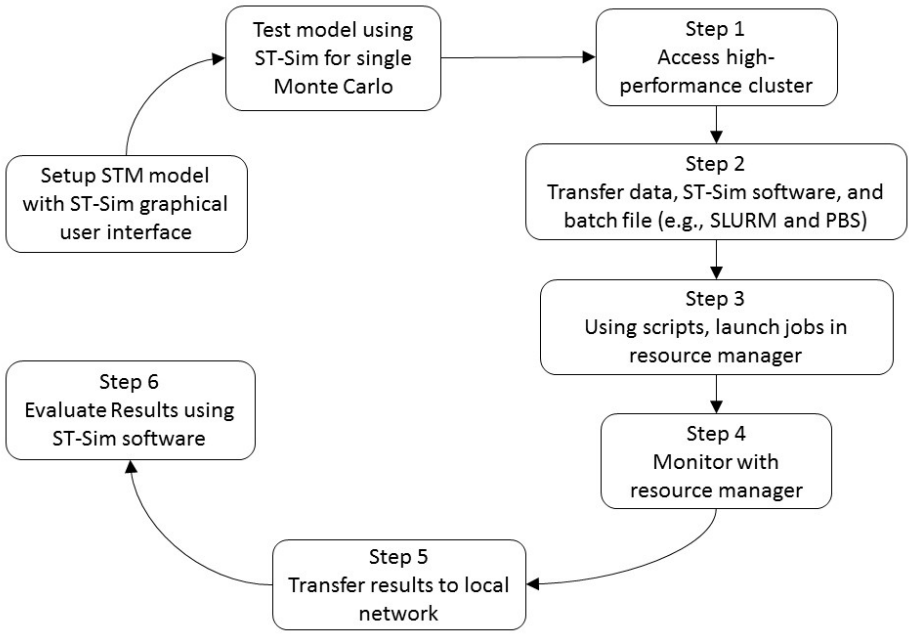

Figure 6. A high-performance computing workflow with state-and-transition simulation models. Simple Linux Utility Resource Manager (SLURM), like the Portable Batch System (PBS), is middleware used to launch and manage tasks in HPC environments. This workflow is very similar to Figure 3, but an HPC system does require a resource manager to distribute the tasks to different compute nodes.

3.7. Monitoring and reviewing embarrassingly parallel tasks

I used two different software packages for monitoring the embarrassingly parallel STSM tasks. CycleServer® is free software developed by Cycle Computing® [44] and it monitors historic/current jobs, produces reports on resource use, evaluates the health of the HTC system, deploys software and jobs, and it provides many other functions. I used the open source Ganglia™ [45] to monitor the tasks executed on the HPC Linux cluster, which is a similar system to CycleServer in that it can monitor grids, but it can also monitor HPC clusters. Regardless of the monitoring resources available to the researcher, having access to a tool or command line queries is beneficial for monitoring jobs running within HTC and HPC resource managers.

After developing the workflows and software for executing embarrassingly parallel Monte Carlo simulations of STSMs in HTC and HPC environments, I used the SyncroSim ST-Sim GUI to evaluate the results (Figure 7). I then assessed the merged results to determine if the appropriate time-stamped GIS data and summary statistics existed in the STSM library. Because the ST-Sim GUI does not run on Mono, a researcher must transfer results derived on a non-Microsoft operating system to a native Microsoft environment. Using SAMBA [46] or similar software to read data not using Microsoft Common Internet File System (CIFs) is another method that allows direct access to the data with the ST-Sim GUI. Data transfers are expensive and time-consuming and this is important to recognize and incorporate into operating costs and project deadlines.

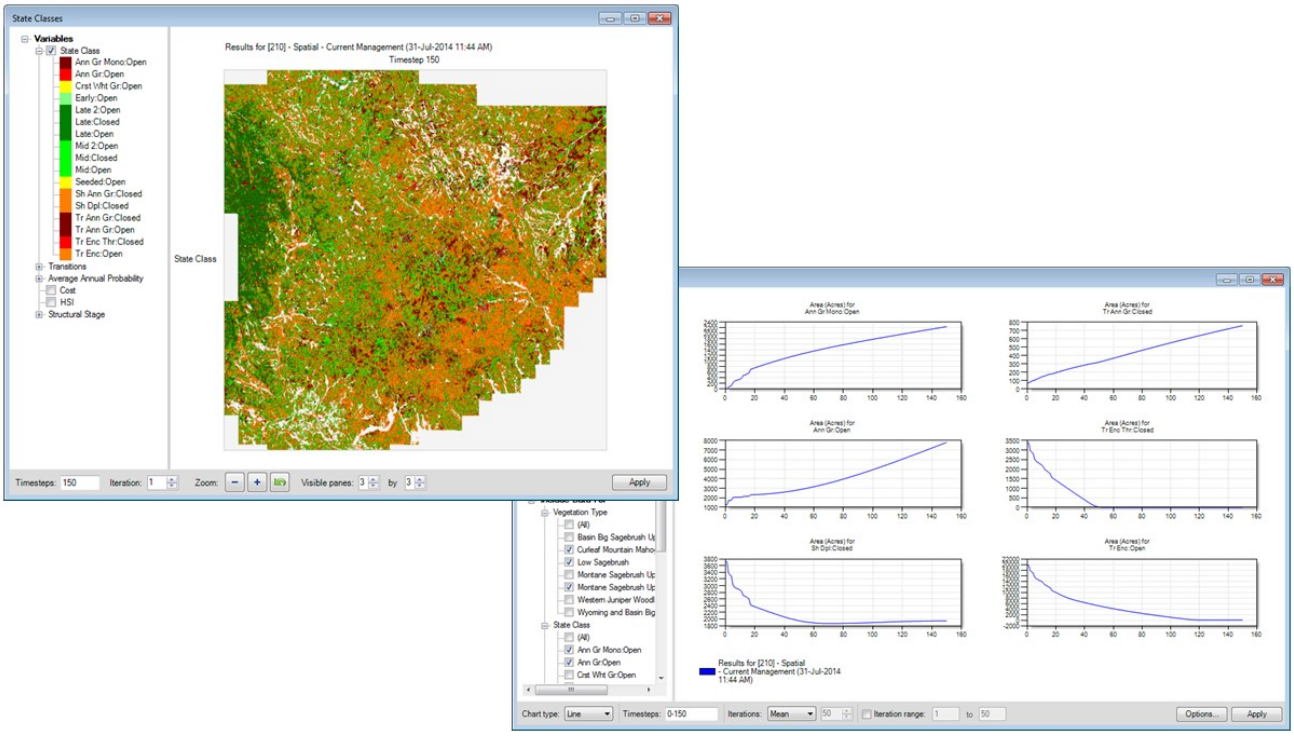

Figure 7. Evaluating the output from the Monte Carlo simulations is important to ensure all simulations completed and the results from the model are reasonable. I used the SyncroSim ST-Sim software package and assessed the spatial and statistical results after the parallel processing completed, as illustrated in the figure.

Numerous metrics exist for examining how well an application implements parallel computing, which in turn provide insight into improving code, identifying problems with hardware configuration, and identifying bottlenecks. These performance indices quantify computing times with respect to similar data and applications, but with different scaled architectures. Several commonly used measurements include the speedup curve, elapsed time, price per performance, speedup, and efficiency [47], and sizeup and scaleup [48]. Like many HTC systems, the environments typically are heterogeneous so Zhang and Yan [49] provide several suggestions of how to quantify performance on heterogeneous environments. Several important factors affect how one accurately measures performance. For example, latency, the amount of time for one computer to connect to another

computer, will affect computations and therefore, runtime. Reading and writing data to different storage types will also affect performance and quantifying performance between different systems can mislead users. Comparing the runtime across different heterogeneous systems is a complex task, and I wanted to track the entire workflow, and not just the time to run a single task or algorithm.

To compare the runtime of the model runs in each computing environment, I measured the elapsed time for the entire workflow. Multiple runs of the same data and software within each computing environment were not evaluated. However, the variability is likely small because in all cases these runs were executed during opportune times with no competition for resources (network traffic or hardware). I measured the runtime (elapsed wall/clock time) for a single replication and 50 simulations on a Windows desktop environment (multiprocessor machine). For HTC, I measured the runtime by how long processing took from the start of the Python Script 1, completion of all Monte Carlo simulations, and the merging of the results. For HPC, I measured the elapsed time for execution of the SLURM script. I also measured the length of time to transfer the data to-and-from the cluster, but I did not include these in the runtime comparisons. Because the hardware (processor speeds and storage) varies slightly between the computing environments, the runtime can only suggest the relative benefits of using parallel computing with STSM.

4. Results

As expected, the results from testing the different combinations of computing methods indicated significant advantages of parallel computing (local, HPC and HTC) over standard, non-parallel methods. The model runs did not encounter errors due to decomposing the STSMs, running the Monte Carlo simulations, merging the Monte Carlo simulation results, or transferring data to-and-from the cluster. In other words, failed jobs due to hardware/network configurations did not influence the elapsed time with any of the tested computing environments. Figure 8 illustrates the computing runtime results from the testing of STSMs based on combinations of hardware, software, and parallel computing.

Several limitations of these runtime comparisons exist. First, HTC is usually a heterogeneous system, while HPC is homogeneous and therefore the hardware was not identical. However, the hardware and operating systems within each system was similar for this study. The HPC system had the greatest variability with processor speeds, while the HTC system did not—in most cases, this is reversed. Second, the storage input/output performance is slightly different between the HTC and the HPC systems. Both systems use NAS devices, but the read/write speeds are slightly different. Third, the transfer of files to-and-from the HPC increases the project time (not included in the runtime; Figure 8) because the U.S. Geological Survey manages the HTC locally and the HPC remotely. Local, HTC, and HPC compute environments behave the same for STSMs because all three use embarrassingly parallel tasks. Because of the architectural (infrastructural) differences between HTC and HPC, I anticipated small differences (e.g., minutes) between runtime.

The speedup of decomposing a problem and concurrently running these individual tasks has an obvious advantage. If a model has 50 Monte Carlo simulations and each Monte Carlo simulation executes on a different processor, then in theory the model could run 50 times faster. With this known advantage, this research did not test the variability of the runtimes within each computing environment. If this study tested the variability, I could have provided confidence intervals around each run time and evaluated the statistical significance of the differences among the computing environments. Based on

the runtime results and that testing was not influenced by competition for computing resources, the variability was expected to be small within each computing environment. When a computing task can be decomposed into equal parts (each Monte Carlo simulation uses the same software and the same data), one expects the computing time to be equally faster. The focus here identified how one can decompose a STSM, identify what computing environments support running the decomposed tasks, and how much overhead is expected for the decomposition and aggregation of the results.

The first computing environment, GUI baseline, required 66 minutes to execute a single replication using the ST-Sim GUI on a local desktop (see Figure 8). If all hardware were equal, that is no bottlenecks existed for compute node interconnects, similar storage read/write speeds, and similar overhead with using middleware for HTC and HPC, then each result would indicate a runtime of nearly 66 minutes. Because the runtimes included the decomposition of the state transition models and merging of the results, causing additional steps, I expected there to be additional overhead. Since these additional steps existed for the parallel (local), HTC, and HPC environments, this will not affect their relative runtime results. I tested the difference in compute time of a single Monte Carlo simulation using a Microsoft Windows desktop (see Table 1, Single desktop[1]) and the command line in its native .NET builds. I then compared this runtime to using a Microsoft Windows desktop (see Table 1, Single desktop[1]) and the command line built with Mono. The difference between the software builds, Mono versus .NET, did not significantly influence the runtime.

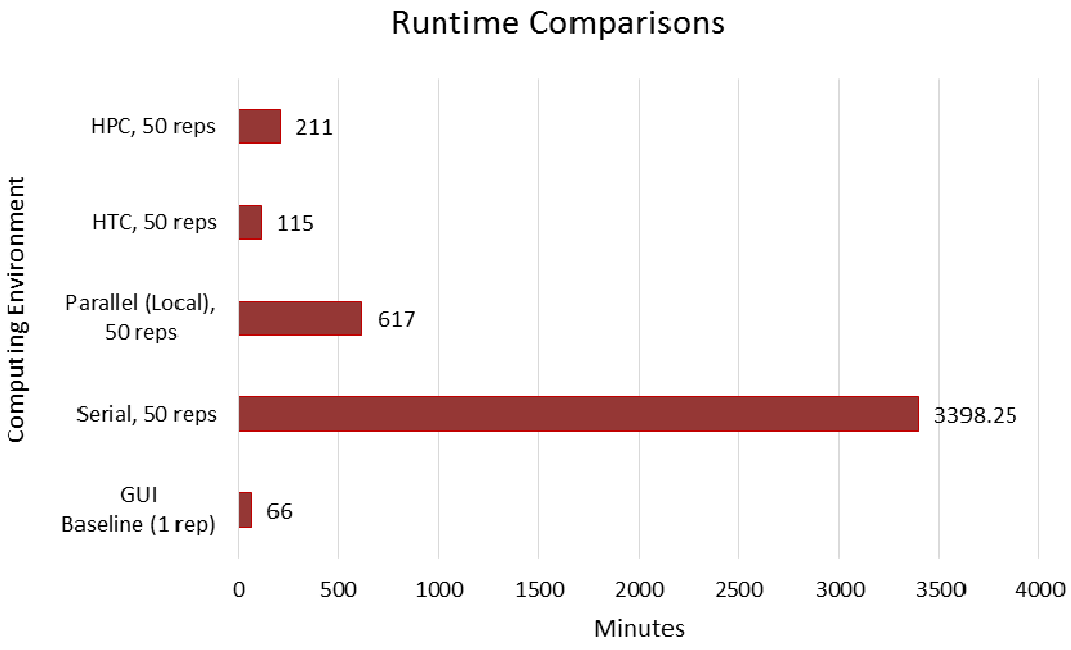

Figure 8. The results of the runtime comparisons for the different computing environments show significant benefits to using embarrassingly parallel tasks for state-and-transition simulation models. The runtime minutes (reported on the x-axis) reflect the elapsed time recorded to run a given number of simulations for each computing environment. Table 1 provides a detailed comparison of each computing environment. Although not included in the HPC runtime presented here, 1.5 minutes were required to transfer the data input to the cluster (0.3 GB) and 24.25 hours to transfer the results back to the local environment (146 GB).

The runtime results indicated significant decreases in computing time for all environments when using embarrassingly parallel tasks. As indicated from figure 8, the computing environments that executed parallel computations were all significantly faster than running state-and-transition simulation models in serial, as expected. However, the embarrassingly parallel tasks of the Monte Carlo simulations also did not complete within the 66 minutes required for a single replication. The HTC environment resulted in a 96.6% reduction of computing time relative to running 50 serial Monte Carlo simulations. The HPC environment had similar results with a 93.8% reduction of computing time relative to running 50 serial Monte Carlo simulations. Although the local parallel test resulted in the smallest reduction of computing time, 81.8%, this result is promising because researchers can significantly improve their workflow with a modest desktop. Parallel computing on the local machine was not as fast as the HTC and HPC environments because the number of available cores was less than the number of Monte Carlo simulations. I purposely designed this scenario to demonstrate the effect of scalability. All parallel computing tasks required greater than 66 minutes, which indicated that the decomposition and merging of the STSMs added an additional cost to the computing time. Based on the recorded elapsed times, HPC required an additional 96 minutes relative to HTC and the HTC runtime resulted in approximately 49 additional minutes relative to the runtime of a single replication. Using HTC and HPC will benefit researchers if they have many Monte Carlo simulations, state-and-transition scenarios, regional data or high-resolution GIS data. This baseline runtime can help researchers identify the benefits of HTC and HPC, and specifically the use of embarrassingly parallel program designs of STSMs.

5. Discussion

Parameterizing state-and-transition simulation models is a complex process and the primary focus for researchers. However, when computing resources become a bottleneck, the following, simple, workflow will help individuals decide what computing resources are necessary while maintaining a focus on building their models. First (1), start small and test your models on a subset of your data and on a local machine. Next, (2), consider testing the model without the GIS data and then slowly building components and complexity to the model. Third (3), if the study area is large, consider dividing the data into ecoregions or meaningful land units. (4) Identify bottlenecks, such as hardware, storage, network, and model complexity by running a single replication locally and compare this runtime to running a single replication with data stored on the network. Also (5), examine memory, disk storage and bandwidth requirements. Usually 1 GB bandwidth is sufficient and unless a researcher plans to use a HPC system, it is unlikely they will have access to faster networks. Finally (6), scale this information to obtain the necessary resources required for the analyses using the fully parameterized models. For example, if you have abundantly more simulations and scenarios than cores, then consider using HTC, a cluster, or one of these systems in the cloud environment.

If you decide to use HTC, build the appropriate files for deploying the simulations using the provided scripts (see Supplementary for software resources), and modify as needed for your system. Start with submitting a single replication to ensure everything is set up correctly which avoids consuming unnecessary computing resources, and then expand to a production level effort. When using HPC, first test the models on a local Linux system before migrating to a cluster. For example, the researcher can build a local Linux box (physical machine) or build a virtual system using Oracle Virtualbox™ (freeware [50]) or VMware®. Once a researcher establishes a viable workflow that

successfully executes on a single Monte Carlo simulation, they can then expand to a production environment and launch all simulations. Small jobs can require longer runtimes in HTC and HPC because of the overhead (e.g., middleware interactions between users and middleware management of computers [inter-communication], decomposition of small jobs, queue lengths, network communications), and therefore there is some trial and error, and expertise, required for determining an optimal workflow.

Running jobs in distributed computing environments can result in failures due to multiple reasons. These reasons can include incorrect parameter definitions, data transfer failures, software and hardware issues, or because users or network activity interrupts the job. The latter scenario typically occurs only when the configuration of the HTC system allows user activity on a networked client to kill the parallel computing application and/or when the system does not use dedicated compute nodes. HTC is designed to handle many tasks that run within a relatively short timeframe. Although executing individual jobs that run longer than 12 hours is possible, such jobs are less desirable because the system is likely to have interruptions from various sources as processing times increase. Many organizations use HTC on dedicated clusters, and in such cases addressing interruptions and configurations are significantly simpler. DAGs can document which jobs fail and allow the user to re-run only those failed jobs. When DAGs are not used, the user must review all log files, determine which jobs failed, and then re-submit these jobs. Using DAGs to track progress reduces overhead and improves workflows, which is why I used them for submitting the STSMs. However, DAGs in general are useful for establishing parent-child relationships between tasks, adding capabilities with pre- and post-scripts, and throttling the number of tasks that are allowed to run in parallel.

Failures with accessing and transferring data, hardware, and other reasons (similar to HTC) also occur on clusters that depend on network infrastructure. However, because an HPC system is mostly homogeneous with respect to operating systems and hardware, and computing nodes are not subject to independent users and uses, failures are less frequent. The primary requirement for using HTC and HPC are that the software running on these systems must properly function with command lines and all messaging and error handling is directed to standard error and standard output versus a GUI message box. A GUI message box requires the user to interact with the software and this is problematic when running many tasks on remote compute nodes. In other words, any software that a researcher wants to use with HTC and HPC cannot require the user to interact with it during any stage. Although exceptions apply and it is possible to navigate such circumstances programmatically using sendkeys (i.e., using code that instructs the software to click a button), these are inefficient and not recommended. In such cases, a researcher should change the software code. During this study, SyncroSim ST-Sim was not properly handling the “stderr” and “stdout” for the command line component, and as a result, when an error occurred the resource managers erroneously considered the tasks as successfully completed; this issue was addressed by ApexRMS and should not cause future problems. Not all software is compatible with distributed computing environments, so researchers should investigate software that they plan on using, their project and modeling workflows, and their anticipated ecoinformatics needs when planning projects and analyses.

An important, additional factor to recognize when using remote hardware is the transfer of data (inputs and outputs) across the WAN. As demonstrated in this study, transferring the results (146 GB) back to the user-environment required considerable, additional time (24.25 hours). The data must be transferred to-and-from remote locations when relying on remote hardware such as remote grids, clusters, and clouds, and these transfers depend on the capacity of the bandwidth. However, when

researchers lack the availability of HTC and HPC systems in a local work environment, utilizing networked resources offers the best option. For smaller data sets, data can be transferred across the WAN; however in most cases, researchers that require HTC and HPC also work with large data sets that cannot be readily and repeatedly transferred across the WAN due to service timeouts, bandwidth limitations (the WAN bandwidth is significantly less than that found on a LAN), and network failures (e.g., dropped packets). Three options for handling larger data sets include permanently hosting the data on remote hardware and shipping the data inputs, outputs on an external storage device, and using Globus™ (http://www.globus.org). These are important considerations because the cost of operations and the time to complete the research may be affected.

Cloud computing expands computing resources through virtualization of hardware and software. Although some organizations rely on private clouds (e.g., NASA Nebula [51]), many publicly accessible clouds exist (e.g., Amazon EC2® [52]; Microsoft Windows Azure® [53]; and Google AppEngine® [54]). Clouds are scalable, increase efficiencies, and decrease hardware costs, but these resources are not necessarily a solution for every application. A researcher using the cloud can select their operating system, as well as, set up HTC and HPC environments. Describing the use of cloud computing extends beyond the scope of this project but a significant amount of literature exists [55]. All storage, proprietary operating systems (e.g., Microsoft Windows®), memory, usage time, and data transfers have associated costs to using the cloud and therefore the project budget, and the economic differences of using the cloud versus other resources may not be clear for every situation [56,57,58]. Regardless of project-to-project differences, the cloud is a viable solution to run SyncroSim ST-Sim as embarrassingly parallel tasks when computing resources not otherwise accessible. High-throughput computing systems, HPC, and computing clusters (networked CPUs) exist at most universities and researchers collaborating with these organizations will often have access (e.g., Open Science Grid, Extreme Science and Engineering Discovery Environment, and Oak Ridge National Laboratory) to computing options. Government agencies may also have access to similar resources. With the basic workflows outlined in this research, the available hardware resources, and updates to the ST-Sim software by ApexRMS, researchers can seek alternative computing resources when required.

Additional modifications to ST-Sim software may increase efficiencies and reduce computing time. The command line version of ST-Sim, and its ability to decompose the Monte Carlo simulations into individual models, was a significant enhancement that supports the use of modeling in HTC and cluster environments. The version of ST-Sim software used in this study required read/write of ASCII files followed by conversion of these file formats to GeoTiff format. Instead, the software could use the Geospatial Data Abstract Library (GDAL [59]) to read/write directly to GeoTiff files. This would reduce the number of data conversions and should further reduce the overall runtime of the STSMs, but this was not explicitly tested here. When using centralized storage, ST-Sim does not need to duplicate the raster data inputs during the decomposition process. However, if the simulation requires sending data to each client and for each Monte Carlo simulation, then duplicating the raster data to the remote locations is necessary. Greater control of this workflow could allow for a reduction in storage requirements by avoiding the duplication of data and minimizing data transfers. Because ST-Sim uses a SQLite database, copies of the database are necessary because these databases do not support asynchronous writes [60]. The SyncroSim ST-Sim software used in this study did not support the "advanced configuration" to average the annual transition probability for specified time-steps because each simulation runs independently of each other and generating these results cannot occur until after the merging of the simulations. However, the functionality does exist in the GUI application. Having

this capability with the command line software would allow researchers to evaluate these results, which would eliminate the need to copy all data from a remote location during the evaluation of the results. Although these are minor enhancements, these changes could reduce runtime and help mitigate data storage requirements for larger projects.

Different data and models will yield different results with respect to absolute runtimes. However, the model and data are identical between Monte Carlo simulations for a given model and if you divide these simulations up, they will run faster relative to running them in serial. Because in every STSM, with respect to using embarrassingly parallel computations of Monte Carlo simulations, the runtime will scale linearly and reflect advantages similar to the results, found in this research, assuming that machines are identical and nothing is interfering with individual jobs. Therefore, the results of this research will occur with other STSMs and data, and the speedup time will depend on the number of Monte Carlo simulations that can run concurrently.

The results of the state-and-transition simulation model using embarrassingly parallel computing will always be as accurate as the model itself. The decomposition uses the exact same data and models, while only changing the random seed for each Monte Carlo simulation. The results would only be invalid if the underlying model was invalid and the approach to decomposition of the simulations has no effect on the validity or accuracy of the models. Therefore, using data decomposition of the Monte Carlo simulations and running these as embarrassingly parallel tasks on different cores will have no effect on the results of the models.

Decomposing Monte Carlo simulations for state-and-transition simulation models may not adequately reduce the computation time for a researcher when the bottleneck occurs because of the runtime during a single Monte Carlo simulation. For the aforementioned reasons of state-and-transition simulation model complexities, a researcher has several options. They can divide the landscape into meaningful units, such as watersheds and ecoregions, process individual strata that define the model and a combination of a single model (often defined as different management solutions) scenario and stratum. If these methods continue to result in long or unacceptable runtime for a single Monte Carlo simulation, then the researcher could work directly with ApexRMS to work out MPI applications. MPI is difficult and expensive to implement, especially with respect to the already inherent complexities of STSMs and spatial dependencies of neighboring pixels within some models.

6. Conclusion

Parallel computing of state-and-transition simulation models is instrumental for research that relies on complex models, large landscapes, and/or small grain (resolution) spatial data. Ecoinformatics, such as applying parallel computing to investigate complex ecosystem processes, is advantageous because it highly scalable and instrumental for modeling large landscapes. Additionally, it reduces the turnaround time to obtain scientific results for any model, it allows thorough analyses with sufficient parameter sweeps, and it accommodates an appropriate number of Monte Carlo simulations. The results from using embarrassingly parallel computations demonstrated reduced computation time for local, HTC, and HPC environments. However, when the number of tasks exceed the number of cores on a local environment, HTC and HPC were more appropriate. Loosely coupled systems (e.g., grid, HTC, or HPC) scale better than tightly coupled systems (e.g., single computer with multiple cores), which is why distributed computing performed best in these evaluations. Because horizontal scaling can use commodity (low cost) hardware and because these systems scale larger than

what is possible with many vertically scaled systems, they are extremely beneficial for larger jobs with many tasks. Because I used embarrassingly parallel computations and data decomposition, high-throughput computing was considered the most appropriate. However, when clusters that support high-performance computing are available, they also can execute embarrassingly parallel tasks. Importantly, I did not change the code of SyncroSim ST-Sim by using functional decomposition, but rather I decomposed a large problem and distributed these tasks (Monte Carlo simulations) to run independently on different computers.

The runtime results of this study highlight several important considerations. Although using HTC and HPC speeds up the analyses, processing time for individual tasks (e.g., a single Monte Carlo simulation) will not be faster. Embarrassingly parallel *design* is not a parallel *function* and it does not decompose a single Monte Carlo simulation using MPI. However, distributing the workload has great benefits, and it is a simple approach to addressing complex STSMs. If all hardware, network, and resource managers are equal the runtime for HTC and HPC would also be equal. However, my results showed that an additional 96 minutes was required for HPC compared to HTC running the same tasks. This occurred because the processors available on the HTC system were nearly identical (Xeon 3.2 GHz) and the processors on the HPC systems were variable and of lesser quality (ranged between 2.7–3.0 GHz), which may be similary to many, existing networked systems. In addition, a single replication completed in about 56 minutes, but HTC required 115 minutes to execute 50 simulations in parallel. The additional 59 minutes were used for decomposing the data, submitting the individual Monte Carlo simulation tasks to the clients, managing the tasks with the resource manager, and merging the resulting runs into a final ST-Sim database (with each of the 50 simulations run simultaneously and independently during this time). Therefore, smaller jobs may run faster without using embarrassingly parallel tasks (when the time for additional simulations is less than the approximately 60 minutes required for process management), and these tradoffs demonstrate that experience and some trial and error are necessary to develop research workflows.

This research accomplished numerous objectives that provide scientists and managers with alternative methods for using complex STSMs and large data sets. I started by identifying challenges inherent in state-and-transition modeling, and I then identified computing resources, configurations and potential bottlenecks that affect the ability of researchers to complete projects within reasonable timeframes. I discussed the different methods for decomposing STSMs and the methods used in this research. I also described the basics of using parallel computing, high-throughput computing, high-performance computing, and cloud computing with STSMs by decomposing the Monte Carlo simulations into embarrassingly parallel tasks. These results indicate significant advantages to using HTC and HPC for complex STSMs with large data, and with the framework developed from this research, suggesting that scientists may have access to resources to easily expand their STSMs to larger landscapes or expand their model complexities without having to change their research.

Acknowledgments

This work would not have been possible without the efforts of the ApexRMS team (Leonardo Frid, Colin Daniel, and Alex Embry), who released a new version of the ST-Sim SyncroSim software that supports running state-and-transition simulation models via command line within a Microsoft Windows and Mono Linux environment. The ApexRMS team also provided the sample data to use as a case study for executing the simulations in a grid-computing environment, which allowed this effort

to focus on examining the benefits of using distributed computing with state-and-transition simulation modeling efforts.

Conflict of Interest

Any use of trade, firm, or product names is for descriptive purposes only and does not imply endorsement by the U.S. Government.

Supplementary

Any researcher that desires to explore the software scripts developed for this study (Table S1) can find these posted here. In most cases, users will need to modify these scripts because high-throughput and high-performance computing environments use different resource managers (e.g., HTCondor™, PBS, OpenPBS, SLURM) and they are each managed differently and therefore require different methods for submitting jobs. However, these scripts provide an excellent starting point for those not familiar with such resources. The scripts are both open source and freeware and the U.S. Geological Survey maintains no limitations on how these are used or distributed.

Table S1. Descriptions of software/scripts used for running embarrassingly parallel Monte Carlo simulations on a grid and cluster environment. These files are templates that other researchers can use and modify for their applications.

File Name	Computing Environment	Resource Manager	Description
_PyInstaller_STM_runScenario.bat	Grid	HTCondor	This provides an example for compiling Python code using the open source PyInstaller™
STM_runScenario.exe	Grid	HTCondor	Executable compiled from STM_runScenario.py with PyInstaller. This program executes on each client within the HTC system
STM_runScenario.py	Grid	HTCondor	The Python source code (Python Script 2) that executes on each client within the HTC system for processing state-and-transition models
ST-Sim_HTCondor_setup.py	Grid	HTCondor	A Python script (Python Script 1) that generates HTCondor files, decomposes state-and-transition models, and other tasks.
SubmitTemplate.sub	Grid	HTCondor	An HTCondor template file used by ST-Sim_HTCondor_setup.py to guide researchers with using different administrative configurations of HTCondor.
stm.slurm	Cluster	SLURM	A text file template, which is converted to an executable text file, that researchers can use as a guideline for running state-and-transition models with SLURM resource manager. One can also convert these to other resource managers as long as the basic workflow is followed.

Software disclaimer

Although this program has been used by the U.S. Geological Survey (USGS), no warranty, expressed or implied, is made by the USGS or the U.S. Government as to the accuracy and functioning of the program and related program material nor shall the fact of distribution constitute any such warranty, and no responsibility is assumed by the USGS in connection therewith.

References

1. Bestelmeyer BT, Moseley K, Shaver PL, et al. (2010) Practical guidance for developing state-and-transition models. *Rangelands* 32: 23-30.
2. Frid L, Hanna D, Korb N, et al. (2013) Evaluating alternative weed management strategies for three Montana landscapes. *Invasive Plant Sci Manag* 6: 48-59.
3. Costanza JK, Terando AJ, Mckerrow AJ, et al. (2015) Modeling climate change, urbanization, and fire effects on Pinus palustris ecosystems of the southeastern U.S. *J Environ Manage* 151: 186-199.
4. Halofsky JE, Hemstrom MA., Conklin DR, et al. (2013) Assessing potential climate change effects on vegetation using a linked model approach. *Ecol Modell* 266: 131-143.
5. Wilson T, Sleeter B, Sleeter R, et al. (2014) Land-use threats and protected areas: a scenario-based, landscape level approach. *Land* 3: 362-389.
6. Daniel CJ, Frid L. Predicting landscape vegetation dynamics using state-and-transition simulation models. Proceedings of the First Landscape State-and-Transition Simulation Modeling Conference, June 14-16, 2011. 2012. p. 5-22.
7. Booker K, Huntsinger L, Bartolome JW, et al. (2013) What can ecological science tell us about opportunities for carbon sequestration on arid rangelands in the United States? *Glob Environ Chang* 23: 240-51.
8. Bagchi S, Briske DD, Wu XB, et al. (2012) Empirical assessment of state-and-transition models with a long-term vegetation record from the Sonoran Desert. *Ecol Appl* 22: 400-411.
9. Creutzburg MK, Halofsky JS, Hemstrom MA. Using state-and-transition models to project cheatgrass and juniper invasion in southeastern Oregon sagebrush steppe. Proceedings of the First Landscape State-and-Transition Simulation Modeling Conference, June 14-16, 2011. 2012. p.73-84.
10. Strand EK, Vierling LA, Bunting SC (2009) A spatially explicit model to predict future landscape composition of aspen woodlands under various management scenarios. *Ecol Modell* 220: 175-191.
11. Chambers JC, Bradley BA, Brown CS, et al. (2014) Resilience to stress and disturbance, and resistance to Bromus tectorum L. invasion in cold desert shrublands of western North America. *Ecosystems* 17: 360-375.
12. Steele CM, Bestelmeyer BT, Burkett LM, et al. (2012) Spatially explicit representation of state-and-transition models. *Rangel Ecol Manag* 65: 213-222.
13. Bestelmeyer BT, Goolsby DP, Archer SR (2011) Spatial perspectives in state-and-transition models: a missing link to land management? *J Appl Ecol* 48: 746-757.
14. ApexRMS. SyncroSim. SyncoSim ST-Sim software. 2014. Available from: http://wiki.syncrosim.com/

15. Evers LB, Miller RF, Doescher PS, et al. (2013) Simulating current successional trajectories in sagebrush ecosystems with multiple disturbances using a state-and-transition modeling framework. *Rangel Ecol Manag* 66: 313-329.
16. Vajda A. Programming Many-Core Chips. 2011th ed. Springer; 2011.
17. Gropp W, Lusk E, Skjellum A. Using MPI: portable parallel programming with the message-passing interface (Scientific and Engineering Computation). 3rd ed. The MIT Press; 2014.
18. Hennessy JL, Patterson DA. Computer architecture: a quantitative approach. 5th ed. Elsevier; 2012.
19. Schauer B (2008) Multicore processors-a necessity. *ProQuest Discovery Guides* 1-14.
20. Chapman MT (2005) The benefits of dual-core processors in high-performance computing. *White Paper* 18.
21. Microsoft Developer Network. Memory limits for Windows releases. Available from: http://msdn.microsoft.com/en-us/library/aa366778.aspx
22. nixCraft. Maximum memory and CPU limitations for Linux server. Available from: http://www.cyberciti.biz/tips/maximum-memory-and-cpu-limitations-for-linux-server.html
23. Flynn MJ (1972) Some computer organizations and their effectiveness. *IEEE Trans Comput* 100: 948-960.
24. Rauber T, Runger G. Parallel programming for multicore and cluster systems. Springer-Verlag, Berlin, Heidelberg; 2010.
25. Darema F. The SPMD Model: Past, Present and Future. In: Cotronis Y, Dongarra J, editors. In Recent Advances in Parallel Virtual Machine and Message Passing Interface. Springer Berlin Heidelberg; 2001. p. 1.
26. Chang C-C, Czajkowski G, Eicken T Von, Kesselman C. Evaluating the Performance Limitations of MPMD Communication. ACM/IEEE SC 1997 Conf. 1997; 1-10.
27. El-Rewini H, Abd-El-Barr M. Advanced computer architecture and parallel processing (Wiley Series on Parallel and Distributed Computing). 1st ed. Wiley-Interscience; 2005.
28. Patterson DA, Hennessy JL. Computer organization and design. 5th ed. Green T, McFadden N, editors. Morgan Kaufmann; 2013.
29. Silva LME, Buyya R. Parallel programming models and paradigms. High Performance Cluster Computing: Programming and Applications, Volume 2. 1st ed. Prentice Hall; 1999. p. 4-27.
30. Navarro CA, Hitschfeld-Kahler N, Mateu L (2014) A survey on parallel computing and its applications in data-parallel problems using GPU architectures. *Commun Comput Phys* 15: 285-329.
31. Center for High Throughput Computing U of W-M. HTCondor™ Version 8.0.1 manual. 2013. Available from: http://research.cs.wisc.edu/htcondor/
32. ApexRMS. Getting started. Sample data, ST-Sim-SpatialSample-V2-1-0. 2014. Available from: http://wiki.syncrosim.com/index.php?title=Getting_Started
33. Karl JW, Laliberte AS, Rango A. Spatial dependence of predictions from image segmentation: a methods to determine appropriate scales for producing land-management information. *Int Arch Photogramm Remote Sens Spat Inf Sci* 2007; XXXVIII (4/C7).
34. Miller RF, Rose JA (1999) Fire history and western juniper encroachment in sagebrush steppe. *J Range Manag* 52: 550-559.

35. Baruch-Mordo S, Evans JS, Severson JP, et al. (2013) Saving sage-grouse from the trees: a proactive solution to reducing a key threat to a candidate species. *Biol Conserv* 167: 233-241.
36. Petersen SL, Stringham TK, Roundy BA. (2009) A process-based application of state-and-transition models: a case study of western juniper (Juniperus occidentalis) encroachment. *Rangel Ecol Manag* 62: 186-192.
37. Bates JD, Sharp RN, Davies KW (2014) Sagebrush steppe recovery after fire varies by development phase of Juniperus occidentalis woodland. *Int J Wildl Fire* 23: 117-130.
38. Rowland MM, Suring LH, Tausch RJ, et al. (2010) Dynamics of western juniper woodland expansion into sagebrush communities in central Oregon. *Communities* 16: 13.
39. Kachergis EJ, Knapp CN, Fernandez-Gimenez ME, et al. (2013) Tools for resilience management: multidisciplinary development of state-and-transition models for northwest Colorado. *Ecol Soc* 18: 39.
40. Xamarin. Mono project. Cross platform, open source .NET framework. 2015. Available from: http://www.mono-project.com/
41. SchedMD. Simple Linux resource utility manager workload manager. Software and Online Help. 2014. Available from: http://slurm.schedmd.com/
42. Tatham S. PuTTY. PuTTY. 2014. Available from: http://www.putty.org/
43. Prikryl M. WinSCP free SFTP, SCP, and FTP client for Windows. WinSCP 5.5.6 released. 2014. Available from: http://winscp.net/eng/index.php
44. CycleComputing. Cycle Computing, Better answers. Faster. CycleServer. 2014. Available from: http://www.cyclecomputing.com/
45. Massie M, Contributors. Ganglia monitoring system. 2014. Available from: http://ganglia.sourceforge.net/
46. Conservancy SF. Samba, Opening Windows to a wider world. 2015. Available from: https://www.samba.org/
47. Karp AH, Flatt HP (1990) Measuring parallel processor performance. *Commun ACM* 33: 539-543.
48. Moturi CA, Maiyo SK (2012) Use of MapReduce for data mining and data optimization on a web portal. *Int J Comput Appl* 56: 39-43.
49. Zhang X, Yan Y. Modeling and characterizing heterogeneous parallel networks computing of workstations of Texas at San Antonio. Parallel and Distributed Processing, 1995 Proceedings Seventh IEEE Symposium on IEEE. 1995. p. 25-34.
50. Oracle. Oracle technology network. Oracle VM VirtualBox. 2014. Available from: http://www.oracle.com/technetwork/server-storage/virtualbox/downloads/index.html
51. National Aeronautics and Space Administration. Open Government Initiative. NASA Nebula Cloud Computing Platform. 2015. Available from: http://www.nasa.gov/open/plan/nebula.html
52. Amazon. Amazon Web Services. Amazon EC2. 2015. Available from: http://aws.amazon.com/ec2/
53. Microsoft. Microsoft Azure. The Cloud for Modern Business. 2015. Available from: http://www.windowsazure.com/en-us/
54. Google. Google Cloud Platform. Google App Engine: Platform as a Service. 2015. Available from: https://developers.google.com/appengine/
55. Eri T, Mahmood Z, Puttini R. Cloud computing: concepts, technology & architecture. Prentice Hall; 2013.

56. Alford T, Morton G. The economics of cloud computing. Booz Allen Hamilton. 2011. Available from: http://broadcast.rackspace.com/hosting_knowledge/whitepapers/Cloudonomics-The_Economics_of_Cloud_Computing.pdf
57. Kondo D, Javadi B, Malecot P, Cappello F, et al. Cost-benefit analysis of cloud computing versus desktop grids. 2009 IEEE International Symposium on Parallel & Distributed Processing. 2009. p. 1-12.
58. Nanath K, Pillai R (2013) A model for cost-benefit analysis of cloud computing. *J Int Technol Inf Manag* 22: 93-118.
59. GDAL. GDAL-Geospatial data abstraction library. Open Source Geospatial Foundation. 2015. Available from: http://www.gdal.org/
60. SQLite. An Asynchronous I/O module for SQLite. 2015. Available from: http://www.sqlite.org/asyncvfs.html

AIMS Press