Computational Cost Analysis of Large-Scale Agent-Based Epidemic Simulations

Tariq Kamal

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in the partial fulfillment of the requirement for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Ali Raza Ashraf Butt
Madhav Vishnu Marathe
Keith R. Bisset
Anil Kumar S. Vullikanti
Martin Schulz

June 05, 2015
Blacksburg, Virginia, USA

Keywords: Computational Epidemiology, Performance Modeling, Load Balancing, Cost Estimation, Distributed Systems

Computational Cost Analysis of Large-Scale Agent-Based Epidemic Simulations

Tariq Kamal

ABSTRACT

Agent-based epidemic simulation (ABES) is a powerful and realistic approach for studying the impacts of disease dynamics and complex interventions on the spread of an infection in the population. Among many ABES systems, EpiSimdemics comes closest to the popular agent-based epidemic simulation systems developed by Eubank, Longini, Ferguson, and Parker. EpiSimdemics is a general framework that can model many reaction-diffusion processes besides the Susceptible-Exposed-Infectious-Recovered (SEIR) models. This model allows the study of complex systems as they interact, thus enabling researchers to model and observe the socio-technical trends and forces. Pandemic planning at the world level requires simulation of over 6 billion agents, where each agent has a unique set of demographics, daily activities, and behaviors. Moreover, the stochastic nature of epidemic models, the uncertainty in the initial conditions, and the variability of reactions require the computation of several replicates of a simulation for a meaningful study. Given the hard timelines to respond, running many replicates (15-25) of several configurations (10-100) (of these compute-heavy simulations) can only be possible on high-performance clusters (HPC). These agent-based epidemic simulations are irregular and show poor execution performance on high-performance clusters due to the evolutionary nature of their workload, large irregular communication and load imbalance.

For increased utilization of HPC clusters, the simulation needs to be scalable. Many challenges arise when improving the performance of agent-based epidemic simulations on high-performance clusters. Firstly, large-scale graph-structured computation is central to the processing of these simulations, where the star-motif quality nodes (natural graphs) create large computational imbalances and communication hotspots. Secondly, the computation is performed by classes of tasks that are separated by global synchronization. The non-overlapping computations cause idle times, which introduce the load balancing and cost estimation challenges. Thirdly, the computation is overlapped with communication, which is

difficult to measure using simple methods, thus making the cost estimation very challenging. Finally, the simulations are iterative and the workload (computation and communication) may change through iterations, as a result introducing load imbalances.

This dissertation focuses on developing a cost estimation model and load balancing schemes to increase the runtime efficiency of agent-based epidemic simulations on high-performance clusters. While developing the cost model and load balancing schemes, we perform the static and dynamic load analysis of such simulations. We also statically quantified the computational and communication workloads in EpiSimdemics. We designed, developed and evaluated a cost model for estimating the execution cost of large-scale parallel agent-based epidemic simulations (and more generally for all constrained producer-consumer parallel algorithms). This cost model uses computational imbalances and communication latencies, and enables the cost estimation of those applications where the computation is performed by classes of tasks, separated by synchronization. It enables the performance analysis of parallel applications by computing its execution times on a number of partitions. Our evaluations show that the model is helpful in performance prediction, resource allocation and evaluation of load balancing schemes. As part of load balancing algorithms, we adopted the Metis library for partitioning bipartite graphs. We have also developed lower-overhead custom schemes called Colocation and MetColoc. We performed an evaluation of Metis, Colocation, and MetColoc. Our analysis showed that the MetColoc schemes gives a performance similar to Metis, but with half the partitioning overhead (runtime and memory). On the other hand, the Colocation scheme achieves a similar performance to Metis on a larger number of partitions, but at extremely lower partitioning overhead. Moreover, the memory requirements of Colocation scheme does not increase as we create more partitions. We have also performed the dynamic load analysis of agent-based epidemic simulations. For this, we studied the individual and joint effects of three disease parameter (transmissiblity, infection period and incubation period). We quantified the effects using an analytical equation with separate constants for SIS, SIR and SI disease models.

The metric that we have developed in this work is useful for cost estimation of constrained producer-consumer algorithms, however, it has some limitations. The applicability of the metric is application, machine and data-specific. In the future, we plan to extend the metric to increase its applicability to a larger set of machine architectures, applications, and datasets.

Dedication

Dedicated to my parents, wife and lovely daughters,
for their endless love, encouragement and support ...

Acknowledgments

I would like to express my deepest gratitude to my mentor, Keith R. Bisset for guiding me through the PhD program. He is an outstanding researcher who shows strong hands-on problem solving ability, in addition to system and software engineering skills. Through his constant presence, he offered insightful advice and useful feedback on papers and presentations. My meetings and discussions with him had laid down the foundation of this research. With his help, I had attended a series of conferences, because of which I met people and stayed at the cutting edge of the literature.

Also, a special special thanks to Madhav V. Marathe for giving me an opportunity to work at NDSSL, and exposing me to the healthy multi-disciplined research environment. It was at NDSSL that I improved my technical skills and grew as an independent researcher. I find his personality and work ethics highly professional. Further, his research ideas have helped me carry my work into the right direction. I have always found him full of hope and encouragement.

I would further like to thank my advisory committee members Ali R. Butt, Anil Vullikanti, and Martin Schulz for their invaluable feedback on my dissertation work. I am particularly thankful to Martin for agreeing to work on my committee and sharing useful research directions. I would also like to extend my thanks to Calvin Ribbens for helping me at many crucial junctions in my PhD studies.

I would like to acknowledge my colleagues at NDSSL, particularly, Jae-seung Yeom, Ashwin Aji, Md Hasan, Harshal Hayatnagarkar, Maksud Alam, and Md Arif for their constructive comments and insightful discussions. We had a lovely time in the lab and enjoyed wonderful coffee breaks. I would like to mention my lovely friends at Virginia Tech, especially, Zakaullah Zahid, Abid Ullah, Karim Akhtar, Naveed Ahmad, Abdul Hafeez, Arshad Madhmood, and Shamim Javed for their unforgettable company. Thank you all for making my stay memorable at Blacksburg. Together, we had plenty of entertaining events and fantastic moments.

I would like to thank my beloved parents for their immeasurable love, encouragement and support. I would like to thank my brothers Shaukat Hayat and Arshad Kamal, for their constant help. The consolation and encouragement from my family has always motivated and invigorated me, and further motivated me to endure and survive the difficulties that I had encountered during my graduate life. I would also like to thank my wife for continuously loving and supporting me through the years. I am grateful to her for taking care of the

household that gave me time to focus on my studies. I would like to thank my lovely daughters, as their love and smiles have kept me striving to achieve my goals.

Last but not the least, I would like to thank the KPK University of Engineering and Technology Peshawar, Pakistan for sponsoring my PhD studies.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CPC Constrained Producer-Consumer

j^{th} task in class C_i $t_{i,j}$

Total messages in all tasks $|M|$

Total incoming messages in all tasks M^+

Total outgoing messages in all tasks M^-

Number of remote messages of a task $R^t_{i,j}$

Number of remote messages of a partition $R^p_{i,j}$

Total remote messages of a class R^c_i

Degree of a task $D^t_{i,j}$

Indegree of a task $D^{t+}_{i,j}$

Outdegree of a task $D^{t-}_{i,j}$

Class i C_i

Degree of a class D^c_i

Indegree of a class D^+_i

Outdegree of a class D^{c-}_i

Total degree of all classes D

Total indegree of all classes D^+

Total outdegree of all classes D^-

Compute load of a partition $L^P_{i,j}$

Compute load of a class L^C_i

Mean compute load of a partition $\overline{L}_{i,j}^{P}$

Mean compute load of a class $\overline{L}_{i}^{C}$

Computation imbalance I_i

Communication load CL

Roun Robin RR

Colocation scheme C

Metis scheme M

MetColoc scheme MC

Chapter 1

Introduction

Pandemic diseases represent one of the most serious threats to public health security [1–4]. In today's interconnected world, an infectious disease can spread very fast, making the public health an important area of research. To adequately respond to such threats, policy makers and health authorities need to study the dense social interactions created by human behaviors and their day-to-day activities.

Developing computational models to study such epidemics is complicated and scientifically challenging. Agent-based epidemic simulation (ABES) consists of epidemic simulations that are based on agent-based modeling. In these simulations, persons are represented as agents. It is a powerful and realistic approach for studying the impacts of disease and population dynamics on the spread of disease when simulating complex interventions. Among many of ABES were the ones developed by Barret et. al [5–9], Eubank et al. [10, 11], Longini et al. [12], Ferguson [13], and Parker et al. [14]. The models permit the study of complex systems as they interact, thus enabling researchers to model the spread of an epidemic and the individual behaviors. However, pandemic planning at the world level requires simulation of over 6 billion agents, where each agent possesses a unique set of demographics, daily activities, and behaviors. The simulation itself is a complex interaction between the spread of the disease, government interventions, and individual reactions. In contrast to many large physics simulations, the outcome of a single run of a socio-technical simulation is not by itself interesting. The stochastic nature of this simulation, the uncertainty in the initial conditions, and the variability of reactions require the computation of several replicates. While, running several replicates of a configuration can help in achieving tighter bounds on the output, simulating many configurations can better explore the parameter space. Given the limited timelines to respond, running many replicates (15 — 25) of several configurations (10 — 100) can be a daunting task.

These compute-heavy and communication-intensive simulations can only be possible on high-performance clusters. The HPC clusters, with their huge computational power makes these large-scale simulations possible: however, at the same time, such computing clusters introduce design and runtime challenges. For efficient utilization of computing resources (i.e., processors), the computation needs to be balanced across all the processors (over the course of simulation), while minimizing the remote communication at the same time. To achieve this, we need better load distribution and resource allocation strategies. The mathematical

formulation of computational and communication workloads is crucial for the design of load distribution algorithms and cost estimation. The cost modeling is also important for gaining insights in order to design future petascale computing systems.

This dissertation, thus, develops a cost estimation model for estimating the cost of agent-based epideimc simulations and presents load distribution schemes to facilitate the efficient utilization of high-performance clusters.

1.1 Challenges in Cost Estimation and Load Distribution

The complexity and irregular nature of agent-based epidemic simulations pose many challenges during the cost estimation and load balancing. In the following few paragraphs, these challenges will be explained in detail.

1.1.1 Large and Irregular Applications

A large-scale graph-structured computation is central to the processing of ABES simulations. The ABES perform variable computation and large communication. The evolutionary nature of their communication and computation introduce load imbalances, which limits the ability of these simulations to achieve higher scalability on high-performance clusters.

The natural graphs [15] commonly used in real-world applications have highly-skewed power-law degree distribution, where a small subset of the vertices connects to a large fraction of the graph. Such graphs are very difficult to partition [16, 17] using simple methods. The in-edges and out-edges (of vertices) play a vital role in the load balancing of such applications on high-performance clusters. The partitioning problems are NP-complete [18], and heuristic solutions have been tried to find near optimum solution.

1.1.2 Split Processing and Computation-Communication Overlap

Processing in ABES is performed by classes of tasks that are separated by global synchronization. The entities have a producer-consumer relationship, where the producer produces messages and signals the consumers to consume them. Since the processing happens on natural graphs, the non-overlapping computations introduce load balancing challenges due to the presence of star-motif nodes. Moreover, each class (of tasks) requires separate load balancing.

Split computation is also important from the cost estimation perspective, as it imposes challenges for such estimation. Further, global synchronization between classes of tasks introduces idle times, which means that the total compute time is not just the summation of compute times on individual processors, but it also requires the accountability of idle times.

Another challenge in ABES is that computation is overlapped with communication. The overlap, if it happens, can make the cost estimation very challenging. The amount of computation-communication overlap is application, data, system, and core count specific, and a general rule cannot be established to capture it in advance.

Another important challenge for the cost estimation models is the determination of relative contribution of computational and communication components in the simulation.

1.1.3 Disease Dynamics and Iterative Nature of Simulation

The ABES systems are iterative in nature. In ABES, the agents interact at locations. These interactions may result in transmission of disease from an infectious to a susceptible individual. A change in the state of an agent may change the list of locations that it will visit in the next iterations. This may in turn change the network structure, and further, the number of computations and communication in the next iteration. This implies the existence of different amounts of workloads (computational and communication) on processors through the iterations. The most significant parameters that inuence the state of an agent are disease transmissibility, incubation period duration, and infectious period duration. The individual effects of these disease model parameters on the computational and communication workloads vary, based on the different disease models (SIR, SIS, and SI disease models: all variations of basic SEIR model). In this work, we do not study the effects on interventions (i.e., agent-isolation etc.) on compute time.

1.2 Research Contributions

This dissertation focuses on developing a cost estimation model and load distribution schemes in order to increase the efficiency of agent-based epidemic simulations on high-performance clusters. While developing the cost model and load distribution schemes, we perform the static and dynamic load analysis computation and communication workloads in such simulations. The static load analysis is performed on the input graph to simulation, and does not consider the load variation (through iterations) from dynamics of disease models. In contrast, the quantification of disease dynamics captures the effects of disease dynamics. In the following section, we highlight the contributions this research would offer through this dissertation.

1. We statically quantify the computational and communication in EpiSimdemics. We determine that the agent carry similar compute loads (assigned a weight of 1), and the location compute load is proportional to the number of its incoming edges.

2. We design, develop and evaluate a cost model for estimating the execution cost of large-scale parallel agent-based epidemic simulations on high-performance clusters. The cost model uses computational imbalances and remote communication to determine a single cost value. We provide a general method for determining their application and machine-specific constants. The model is validated using cross-validation techniques and actual execution times of EpiSimdemics. Our evaluations show that the cost metric estimation of execution times is helpful in performance prediction, resource allocation and evaluation of load distribution schemes.

3. We develop a method to use Metis for partitioning agent-location bipartite graphs to achieve static load balancing in EpiSimdemics. We also develop two lower-overhead custom schemes: Coloc and MetColoc. We evaluate the strong scaling performance of Metis, Colocation and MetColoc. Our analysis shows that MetColoc achieves a performance similar to Metis, but doing so with half the partitioning time (compared to Metis). The Colocation scheme shows a comparable performance to Metis when creating a larger number of partitions. The Colocation partitioning time is very small in comparison to Metis.

4. We study the individual and joint effects of disease model parameters (transmissiblity, infectious period and incubation period) on the compute time in ABES. We quantify the effects using analytical equations for different disease models (SIS, SIR and SI). We also combine the static and dynamic quantification equations to extract a dynamics-aware cost estimation metric.

1.3 Dissertation Organization

The remainder of the dissertation is structured as follows. In Chapter 2, we discuss the related work and background technologies that lay the foundation for current research. In Chapter 3, we statically quantify the computation and communication in EpiSimdemics. In Chapter 4, we develop a cost metric that estimates the cost of constrained producer-consumer algorithms. The cost metric is an extension of the min-max model developed in [19] and we develop a detailed scientific methodology for extracting the cost equation and its application-specific constants. In Chapter 5, we review and develop static load distribution mechanisms to improve the efficiency of ABES in general and EpiSimdemics in particular. We also highlight the performance bottlenecks (of natural graphs) that are limiting the ability of load balancing algorithms to achieve an increased efficiency. In Chapter 6, we study the individual and joint effects of disease model parameters on the compute time of EpiSimdemics in SIS, SIR and SI disease models. The most important parameters that we study include transmissibility,

infectious period, and incubation period. We develop analytical equations that quantify the effects of disease model parameters on compute time. We also combine the static cost estimation metric with equations of disease dynamics dynamic to extract the dynamics-aware cost estimation model. We conclude in Chapter 7 and suggest directions for future work.

Chapter 2

Background and Related Work

This research focuses on three key areas as it aims to improve the efficiency (scalability) of large-scale agent-based epidemic simulations — cost estimation model, static and dynamic load analysis, and data distribution techniques. This section summarizes the prior work that is closely related to these three areas.

2.1 Cost Estimation Models

In this section, we present an overview of the existing cost estimation models.

2.1.1 Min-max Models

The min-max criteria minimize the maximum cost to solution. It is commonly used to define robust solutions [20] for shortest path, spanning tree, task-assignment, min-cut, knapsack, and many other problems. Chien et al. [19] developed a min-max model for the optimal assignment of tasks to processors. The metric minimizes the execution cost of running all the tasks in parallel distributed systems. This metric was used to measure the effectiveness of task-assignment schemes. It tries to minimize the communication overhead given a certain imbalance in computational load. Further, a slightly modified version of the metric was used in many graph partitioning libraries, including Metis [21–23] and Scotch [24].

Chung-Lun et al. [25] developed a min-max criteria to solve the scheduling problem of absolute lateness. These methods try to find an optimal job schedule that can minimize the maximum weight. The problem is that it is NP-complete even for a single-machine case, and strongly NP-complete for a general case. Empirical testing of this method suggested that the performance is asymptotically optimal, as the number of jobs tends to be infinite.

Chris et al. [26] evaluated the min-max clustering principle for measuring load balancing in partitions. They proposed an objective function for graph partitioning that follows the

min-max clustering principle. The theoretical analysis of the min-max cut indicates that it leads to balanced partitions.

Ercal et al. [27, 28] also developed a cost function, wherein the task assignment algorithms try to minimize. The cost function tries to minimize the inter-process communication costs, given that the computational loads are balanced across various processors.

Nikhil et al. [29] studied the graph partitioning problems from a min-max perspective. Their objective was to minimize the maximum number of edges leaving a single part. The two main versions they considered were, firstly, where the k parts need to be of equal size, and secondly, where they must separate a set of k given terminals.

2.1.2 Other Cost Estimation Schemes

Besides the min-max models, there were several other models developed for performing the cost estimation of parallel algorithms. Shen et al. [19] proposed a cost function for evaluating the effectiveness of task assignment. The cost function measures the maximum time taken for a task to be completed. The method was useful for task-scheduling.

Angelia and Asuman [30] presented an analysis of a distributed computation model for optimizing the sum of objective functions in multiple agents. This model requires dynamic updates for cost estimation and is used for the analysis of computation in a distributed environment.

Further, Roig et al. [31] explored the current models used in mapping of parallel programs: Task Precedence Graph (TPG), Task Interaction Graph (TIG) and later, defined a model called Temporal Task Interaction Graph (TTIG). TTIG enhances the previous two models with the ability to explicitly capture the potential degree of parallel execution between adjacent tasks, allowing the development of efficient mapping algorithms. Overall, the model was useful for mapping.

Profiling is a dynamic program analysis method that measures the memory and compute time of codes [32–34]. The analysis can be at the application level or at the functional level. Profiling is particularly useful in adaptive systems such as just-in-time compilers, dynamic optimizers, power optimizers, and binary translators.

2.1.3 Limitations of Previous Work

The existing cost estimation metrics are widely used in their respective settings. However, none of them address the special class of constrained-producer consumer algorithms that we target in this work. Shen's [19] method is useful for task-scheduling. However, this method does not cover the cost analysis of irregular applications. Moreover, Angelia and

Asuman's [30] model also requires dynamic updates for cost estimation, which does not make it practical for static cost analysis. Roig et al. [31] models are useful for mapping as a whole, but fail to produce a comprehensive analysis of simulation costs. Chien et al. [19], Chris et al. [26] and Ercal et al. [27] developed cost functions that the task assignment algorithms try to minimize. However, the models are not directly applicable to applications, where the processing is performed by classes of tasks that are separated by global synchronization. Profiling schemes are good for dynamics analysis of programs and are useful for run time optimizations. However, they do not give you insights about the actual data or algorithm of the application.

2.2 Partitioning and Data Distribution Schemes

Partitioning is an important aspect in distribution of application data and work to processors; this can greatly influence the performance of a parallel simulation. The purpose of partitioning is to maximize the utilization of computational resources and minimize the communication between various processors. While minimizing the interprocess communication tends to assign all the tasks to a single processor, load balancing on the other hand, tries to distribute the computations evenly to all cores. Therefore, conflict exits between these two criteria and a compromise must be made to obtain a policy for optimal assignment. A number of application-specific and general purpose optimization techniques have been developed, which show the benefits in increasing application performance on HPC systems. They can be classified roughly into three categories: (i) Graph-theoretic methods [19, 35–37], (ii) Mathematical programming [38, 39], and (iii) Heuristic methods [40].

2.2.1 Graph-Theoretic Methods

The graph-theoretic methods represent the tasks as a graph, and apply minimal-cut algorithms to achieve the partitions characterized by minimal inter-process communication. Some of the well-known graph theoretic algorithms include Metis [21–23] and Scotch [24]. Metis and Scotch use a cost function that the graph partitioning functions try to minimize. A graph-theoritic method allows the user to specify the maximum allowed computational imbalance. Then the partitioning algorithm tries to minimize edge-cut within that limit. Partitioning is NP-hard and the graph partitioning algorithms approximate an optimal solution to the problem.

Metis algorithms are based on multi-level recursive bisection, multi-level k-way, and multi-constraint partitioning schemes. The partitions have reduced remote communication and balanced computations. Further the multi-level k-way partitioning is preferred over multi-level recursive bisection, as it is faster by a factor of $O(\log k)$, and produces better quality partitions [23] in less time.

INRIA [41] established a complexity result that assesses the difficulty of achieving static load balancing in iterative algorithms and designed practical heuristics that provides efficient distribution schemes.

Cosenza [42] presents a distributed dynamic load balancing approach for achieving load balancing in parallel agent-based simulations. In such simulations, a large number of agents move through the space, obeying some simple rules. Catalyurek [43] presents a new hypergraph model for dynamic cases. This algorithm performs repartitioning at during simulation execution and claims performance numbers comparable to ParMETIS.

2.2.2 Mathematical Programming

The mathematical programming methods formulate the problem as an optimization problem, and solve it with mathematical programming techniques (linear and integer programming). Rao and Chopra [44, 45] developed a linear program to partition the graph for load balancing. Their model minimizes the edge-cut, while keeping the computational load in bound.

2.2.3 Heuristic Methods

The heuristic methods provide fast but suboptimal solutions for task assignment, which are useful for applications where an optimal solution cannot be obtained in a reasonable time. The simplest example is of round-robin distribution. This strategy does not consider the relative weights of tasks and simply assigns them to partitions in a round-robin fashion. Round-robin distribution has no overhead and can be easily done at the start of a simulation. The distribution it provides is usually imbalanced in terms of computations and communication.

2.2.4 Limitations of Previous Work

The graph partitioning algorithms discussed in the previous section have two limitations. Firstly, they are good for the general cases, but they do not address the special class of network-based bipartite graph-structured applications, where the computation happens in phases between classes of nodes. Additional work is required to use these graphs for irregular applications. Secondly, the algorithms are expensive in terms of partitioning time and memory consumption. In fact, the partitioning runtime and memory consumption increase exponentially, as the problem size is increased.

2.3 Disease Dynamics and Graph Dynamical Systems

Graph dynamical systems (GDS) refer to the concept of performing a wide range of activities on graphs or networks and study the resulting global dynamics. It is used to model a large number of complex systems, which are used in sociology, epidemiology, biology and physics [46–49]. The agent-based epidemic simulation can be represented as a GDS. The representation provides a sound basis to develop reaction-diffusion processes, as it enables direct mapping between nodes and edges. The nodes represent the agents, and the edges represent their interactions. The application of the Susceptible-Exposed-Infectious-Recovered (SEIR) model on the person-location graph involves the dynamics of changing network structure and individual behaviors [50–52].

Three of the most important factors that contribute to such dynamics in agent-based epidemic simulations are: (i) disease transmissibility, (ii) infection period duration, and (iii) incubation period duration. The disease parameters affect different disease models differently. There are three popular disease models (SIR, SIS and SIR), which are all variations of the basic SEIR (Susceptible-Exposed-Infectious-Recovered) disease model. Transmissibility refers to transmission intensity of a disease through a population. It is the probability of transmission of infection from an infected individual to a susceptible individual in one minute of contact. The incubation period is the interval during which the infected individuals cannot spread the disease to other individuals. The infectious period duration is the period during which infected individuals can transmit the disease to susceptible individuals.

The epidemic simulations are iterative in nature, and the workload changes with change in the person state due to certain disease dynamics. Quantifying the disease dynamics requires studying the effects of transmissibility, infection period duration and the incubation period duration in the three basic disease models (SIR, SIS and SI). Several studies on disease dynamics and the evaluation of intervention measures have used the same set of transmissibility, incubation period, and infectious period based on the natural history of a disease. Moreover, some studies have explored and evaluated the individual and joint effects of variation in disease model parameters on the simulation dynamics over a wider range. Nsoesie [53, 54], Eubank [55] and Barret [56] performed the sensitivity analysis of agent-based epidemic simulations. Their studies show the effect of disease dynamics on the peak and total attack rate as the sensitivity parameters, transmissibility, infection period duration and incubation period duration are varied. They also show the effects of a network structure in the spread of an infection in a population. However, it doesn't provide any insights on its computational and communication effects. To our knowledge, there has been no such study that measured the impact of disease dynamics on computations in agent-based epidemic simulations.

2.3.1 Limitations of Previous Work

The research conducted by Elaine, Eubank and Barret, studies the effects of transmissibility, infectious period, incubation period and the network structure on the peak and the total number of infections in agent-based epidemic simulations. However, they do not consider the effect of these parameters on the computational cost of the simulation.

2.4 Chapter Summary

In this chapter, we have discussed the previous works related to the cost estimation models, load balancing schemes, and dynamic load analysis. The methods and schemes discussed in the related work are adequate for general cases, but they do not specifically target the more complex case of large-scale agent-based epidemic simulations. The cost estimation model presented in this dissertation targets the cost estimation of constrained producer-consumer algorithms with the main objective of utilizing high-performance clusters in an efficient manner. The metric is applicable to ABES. This dissertation also improves the efficiency of large-scale ABES by using well-known partitioning schemes and semantic-aware data distribution schemes.

Chapter 3

Static Load Analysis

In this section, we statically quantify the computational and communication workloads in EpiSimdemics, using the application semantic and input person-location bipartite graph. We start by giving an introduction of the EpiSimdemics algorithm, and then go into the detail to discuss the quantification of each of the components.

3.1 The EpiSimdemics Algorithm

EpiSimdemics is a popular agent-based parallel simulator, which models the spread of a disease across a social contact network. The network is a bipartite graph of person and location nodes as shown in Figure 3.1. Each edge between a person node and a location node represents a visit to a location by that person. The persons produce messages and send them to locations to consume. The locations compute interactions (e.g., transmission of contagious disease) between all pairs of spatially and temporally co-located people and send the outcomes back to the persons.

A brief description of EpiSimdemics algorithm is given below, and is summarized in Figure 3.2. More details can be found in [5, 9, 57].

During the initialization phase of this simulation, the person and location nodes are statically assigned to chares, which are assigned to processes. A chare in Charm++ [58] is a task decomposition unit that may contain some state (i.e., data), send and receive messages, and perform some computation in response to the receiving of a message. In the next section, we outline the implementation of our algorithm on the Charm++ platform. Further details about our implementation in Charm++ can be found in [6, 59].

After the initialization, the simulation starts, which progresses through several time-steps. The typical interval of a time-step is 24 hours. During each time-step, the simulation performs the following tasks:

1. At the start of each timestep, the person entities compute their schedules. The schedule for a person is the set of all visits (to locations) that the person will make in the current

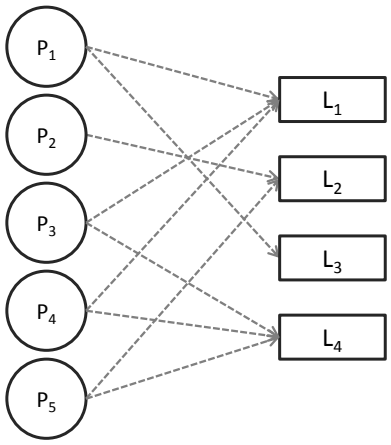


Figure 3.1 A bipartite graph of person and location nodes. P_1, P_2, P_3, P_4, and P_5 represent persons. L_1, L_2, L_3, and L_4 represent locations. The edges represent the communication between person and location nodes.

time-step. For each visit, the person sends a message to its corresponding visited location with the details of the visit (time, duration and health state).

2. The person computation is followed by synchronization. This synchronization guarantees that the locations have received all their visits before starting to process them.

```
1: distribute();                                   ▷ persons and locations to processors
2: initialize();
3: for t = 0 to T increasing by δt  do
4:     foreach person A_j ∈ P_i do                 ▷ send visits to Location tasks
5:         computeVisits();
6:         sendVisits();
7:     end for
8:     Locations ← receiveVisits();
9:     synchronize();
10:    foreach location L_k ∈ P_i do
11:        computeInteraction();                   ▷ Process received messages
12:        sendOutcomes();
13:    end for
14:    Persons ← receiveOutcomes();
15:    synchronize();
16:    foreach A_j ∈ P_i do
17:        updateState();
18:    end for
19: end for
```

Figure 3.2 A pseudocode version of EpiSimdemics algorithm.

3. Locations compute both-way interactions between pairs of visiting persons. If an interaction results in an infection, the location sends a message to the infected person with the information of infection.

4. Similar to person computation, location processing is followed by synchronization to make sure that all the persons have received their infection messages.

5. After the synchronization, persons update their health state based on the infection messages received from locations.

6. Finaly, the global simulation state is updated. After global updates, the next time-step begins.

3.2 EpiSimdemics Implementation

In this section, we describe implementation of the EpiSimdemics algorithm on Charm++ [58]. More details about our implementation in Charm++ can be found in [6, 59]. Charm++ is a distributed task-oriented programming environment, where the parallel program is written in terms of chare objects. The chare objects interact with each other by means of message passing via a remote method invocation. Charm++ offers several exciting features, including optimized communication, improved synchronization, parallel input reading, and computation and communication-aware dynamic load balancers.

In EpiSimdemics implementation [6], we follow a two-level distribution strategy. First, we create the LocationManager (LM) and PersonManager (PM) chare arrays to handle the location objects and person objects respectively, as shown in Figure 3.3. After this creation, we assign the elements of PM and LM chare arrays to processes in a round-robin fashion. Next, we assign person and location objects to the PM and LM chare elements, based on a data distribution strategy. For more details about the distribution strategies, refer to Chapter 5. The individual chares in both the chare arrays handle the computation and communication of their location or person objects respectively. We also use a groupchare array, which stores the global variables — this is updated at the end of each iteration. The chare group instantiates a single chare group object in every process, which is used by other chares (on the same process) to access the resources.

After the initialization is complete, the simulation process is ready to start. At the beginning of each simulation day, the person objects compute their visit messages and send them to the location objects (i.e., the PM chare elements send messages to specific LM chare elements). The location objects compute the infections and send the outcome messages to the person objects. The person objects then decide the locations that they have to visit during the next iteration, and the process continues. The Charm++ runtime system handles the processing of messages at the source and destination chares.

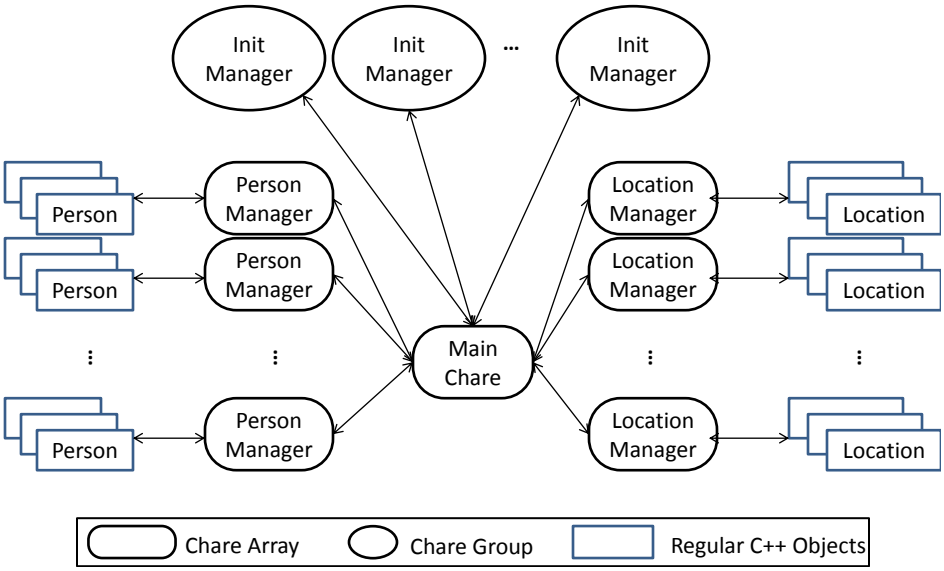


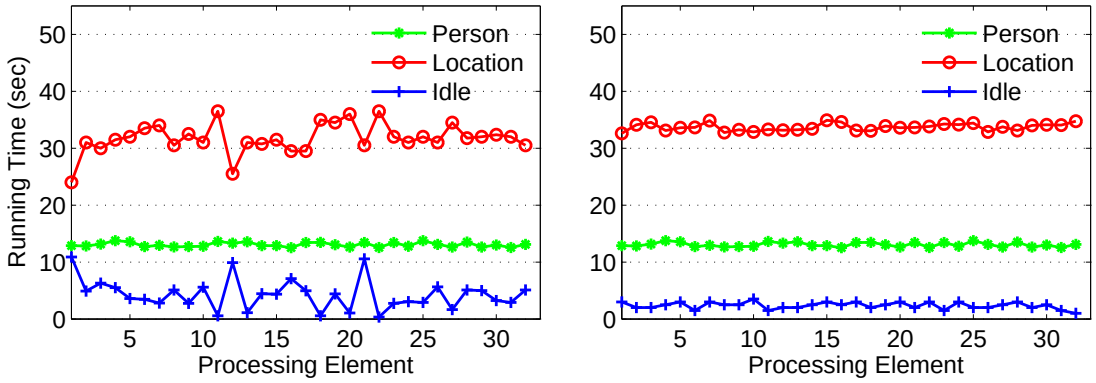
Figure 3.3 Entities in EpiSimdemics.

The simulation has three global synchronization points per iteration. The first is to ensure that every visit message sent from a person object has been received at the destination location, before the computation by that location begins. The second is to ensure that every outcome message sent by a location has been received by the destination person, before the state of the person is updated. The third is to ensure that the global simulation state is updated before the next time-step starts. The first two synchonizations are performed through the Completion Detection (CD) method of Charm++. CD is a method that allows the automatic detection of the completion of a distributed task within a larger application. The completion is reached when the participating chare objects have produced and consumed an equal number of messages. The third synchronization is performed through the contribute call.

At the end of each iteration, the PersonManager chare array elements process the received infection messages and update the person state. After the person's state is updated, and the global variables are computed (to update the simulation state), the next iteration starts.

3.3 Person Computational Load

On an average, a person computation takes 20—25% of simulation execution time as shown by Bisset et al. [6]. In this section, we statically quantify the person computational load in EpiSimdemics. In static quantification, we do not consider the variations in computations and communication resulting from the disease dynamics. EpiSimdemics is an iterative application, wherein the computation and communication workloads may change through iterations as a



(a) Round robin distribution **(b)** Location weighted distribution

Figure 3.4 Per process execution times of persons and locations when simulating the AL data (person nodes: 4,373,206, location nodes: 1,198,947) in EpiSimdemics: (a) Round-robin distribution of person objects to processors gives balanced workloads in terms of person computation. The variation in location compute times shows that the location objects carry variable computational weights. (b) Assignment of location objects to processors based on the mean degree gives the balanced location workloads. Due to the balanced workloads, the idle times are very small.

result of change in the state of the persons. In Chapter 6, we perform a detailed analysis and quantification of disease dynamics.

A person performs two types of computations. Firstly, it produces messages and sends them to locations to consume. On average, each person generates and sends a similar number of messages (5.51 ± 0.18) to locations. Secondly, it consumes the outcome messages produced by locations. The number of messages received from locations are usually negligible. In the SIS model, where the persons can get infected multiple times, and with sufficiently large transmissibility, the number of messages generated by locations are no more than 5% of the total communication volume.

Since the persons generate and send a similar number of messages, we have performed an experiment to see whether the persons also perform a similar number of computations. In the experiment, we assigned persons to 32 processors in a round-robin fashion. We used a synthetic network of Alabama (AL) for this experiment. The data sets were created by our research group using the methods described in [60]. Figure 3.4(a) shows that the round-robin distribution provides balanced workloads in terms of person computation. This confirms that the number of persons assigned to a partition is a good estimation of the person's computational load in that partition.

3.4 Location Computational Load

In this section, we statically quantify the location computational load. Again, we do not consider the variation in the computational load from disease dynamics. Location processing is the most compute-intensive step in EpiSimdemics, and on average, takes 60—70% of the simulation execution time as shown by Bisset et al. [6]. A location performs two types of computations: stores the messages received from persons and processes the received messages.

The first part of location computation involves receiving messages from persons, storing them, and converting them into events. There are two events per person (per visit message): arrive events (person p arrives at location l) and depart events (person p leaves location l). The events are added to arrive and depart event queues, and are sorted by their arrival and departure times respectively.

The second part of location processing involves the computation of interactions between pairs of visiting persons (stored in the arrive queue). A Sequential Discrete Event Simulation (SDES) is performed on the stored events. In SDES, every message creates an arrive event and a depart event, with a timestamp (the arrive/depart time). The events are put in a queue ordered by time. Arrive events add a person to the occupant set. A depart event removes the person from the occupant set and computes the departing person's interaction with everyone in the occupant set, sending outcome messages to anyone infected.

The compute outcome (disease propagation) is modeled by Equation 3.1.

$$p_{i \to j} = 1 - (1 - r_i s_j \rho)^{\tau} \tag{3.1}$$

where $p_{i \to j}$ is the probability of infectious individual i, who is infecting the susceptible individual j, τ is the duration of exposure, r_i is the infectivity of i, s_j is the susceptibility of j, and ρ is the transmissibility, a disease-specific property defined as the probability of a single, completely susceptible person being infected by a single, completely infectious person in one minute of contact.

The interactions performed at a location may result in the transmission of disease from one person to another. In the event of this, the location sends an outcome message to the infected person, along with the information of the outcome. The outcome message contains the person ID, location ID and the time of infection.

The number of such computations performed by a location depends on the number of events generated and the number of interactions performed, which again depends on the number of visit messages received at those locations. Since different locations receive a different number of persons, the number of computations performed by locations is also different. For example, on a normal day, a business location might receive more persons compared to a home location. This means that each of the locations perform a different number of computations. This fact is further illustrated by the imbalanced workloads from the round-robin distribution of locations to the processors as shown in Figure 3.4(a).

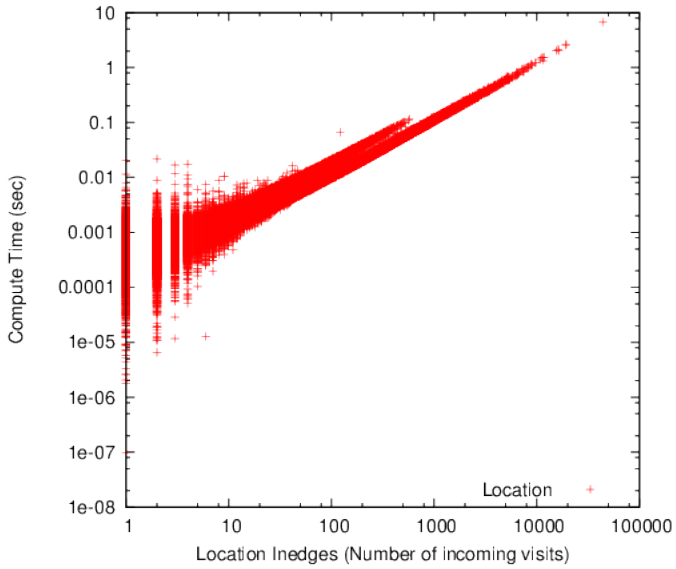


Figure 3.5 Location execution time shows a linear relationship with the number of incoming visits in a location.

Each message sent from a person to a location is a visit to that location by that person. Since interactions must be computed between every pair of spatially and temporally co-located entities, the computation performed by a location task is n^2, where n is the number of visiting persons at that location at any given time. Therefore, the computation performed by a location depends on the number of incoming visits or incoming edges as shown in Figures 3.5. Theoretically, the number of interactions performed by a location is proportional to the square of the incoming visits, but we show that the computational workload of a location can be quantified using the number of visit messages, and not by the square of it. There are two reasons for this conclusion. each location is further divided into one or more sub-locations, where the interactions between persons are performed at the sub-location level (person's are co-located at the sub-location level). Second, the interactions happen between infectious and susceptible persons only (incubatious and removed persons do not interact), thus reducing the number of interaction.

To further emphasize the point that the execution time of a location is proportional to the number of its incoming visits/persons and not the square of it, we have performed an experiment. In this experiment, locations were assigned to processors based on the number of incoming visits. The balanced execution times across processors in Figure 3.4(b) shows that the computation performed by a location is proportional to the number of its incoming visits. Therefore, the number of incoming edges/visits of a location serve as a good measure for the estimation of its computational load.

The computation performed by location is quantified in Equation 3.2.

$$\textit{location computation time} \propto g(M_l^+) \tag{3.2}$$

where $g()$ is the function of location l performed over M_l^+ (the number of its incoming messages).

3.5 Communication Load

The communication in EpiSimdemics happens in two phases. Firstly, at the start of every time-step, the persons produce and send messages to locations. Secondly, the locations send outcome messages to persons. Since the number of outcomes messages is negligible and mostly overlaps with execution, we do not use these messages to estimate the compute time. Moreover, the intra-processor communication shows negligible latency, and we is not used as part of the communication load as well. Therefore, we only measure the communication load in terms of the number of messages that are sent to different processors.

3.6 Chapter Summary

In this chapter, we statically quantified the person, location and communication workloads in EpiSimdemics. The persons perform a similar number of computations, and therefore, their relative computational load on a processor is quantified as the number of persons assigned to it. The locations perform a different number of computations, and accordingly, their load is quantified as the number of incoming visits (of persons) to that location. Finally, the communication load is quantified as the number of inter-process messages sent from persons to locations.

Chapter 4

Cost Estimation Metric

In this section, we develop a cost estimation metric that can quantitatively measure the execution time of agent-based epidemic simulations. The cost metric is an extension to the min-max model developed in [19], and covers the broad class of constrained producer-consumer algorithms. We start with an introduction of the constrained producer-consumer models, and then, discuss the cost estimation metric.

4.1 Constrained Producer-Consumer Algorithms

Producer-consumer-based modeling is a natural and widely used approach for modeling complex systems composed of multiple interacting entities. A large group of parallel algorithms belonging to the class of producer-consumer models can be classified as Constrained Producer-Consumer (CPC) algorithms. The processing in a CPC program is performed iteratively by classes of tasks, $(C_1, \ldots, C_N)$, in N phases as shown in Figure 4.1. The tasks in successive classes have a producer-consumer relationship. In the first $N-1$ phases, tasks in C_k produce messages for tasks in C_{k+1} to consume. In the last phase (phase N), tasks in C_N produce messages for tasks in C_1 to consume. One distinguishing feature of these types of algorithms is that the processing of consumer tasks do not start until they receive all the messages destined for them.

A large number of applications belong to the class of CPC algorithms. Such applications range from modeling genetic algorithms [61–65] and molecular dynamics [66,67], to simulating the outbreak of epidemics [5, 9, 68–70] and the threat of mobile malware [71], from modeling the human immune system [72, 73] to understanding the consumer purchasing behavior [74], besides many others. Besides these applications, many frameworks naturally fit into the CPC category. One such example is the iterative MapReduce [75–77], where the program is run iteratively by sets of mappers and reducers until a certain objective is achieved or the maximum number of iterations is reached. In iterative MapReduce, the mappers and reducers work as producers and consumers alternatively. The processing is separated in time, and the user-defined reducer does not start executing until it receives all the messages sent to it.

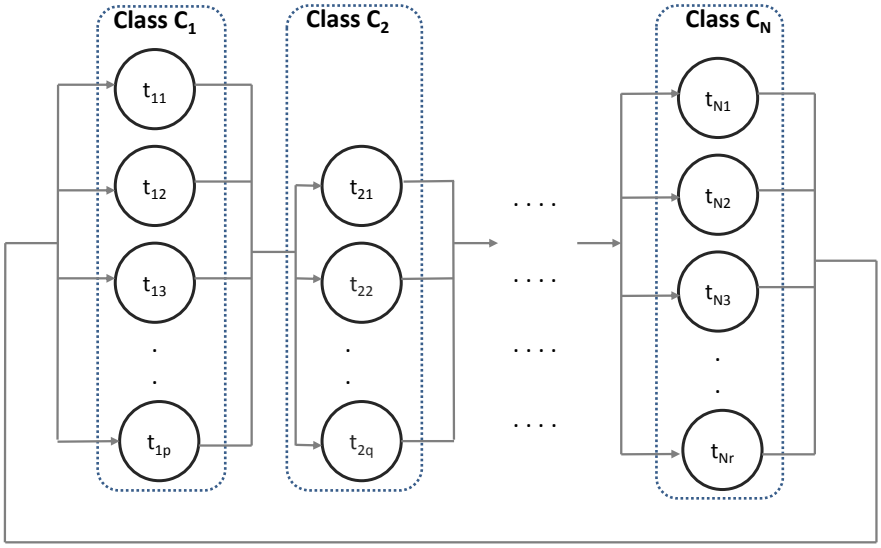


Figure 4.1 Entities in a CPC parallel-program are divided into N classes $(C_1, \ldots, C_N)$. In the bipartite graph, the nodes represent the tasks, and the edges represent the data dependency between those tasks.

4.1.1 General Mathematical Formalization

In this section, we define mathematically formalize the CPC model. A CPC model is defined by an interaction graph, with functions on each vertex of the graph. An example of the interaction graph, $G(V, E)$, is shown in Figure 4.1. In the figure, the vertices represent the tasks, while the edges correspond to communication dependencies between those tasks. Further, each task represents the computation performed in one of the N classes.

The task set is denoted by $v_{i,j} \in V$ where $v_{i,j}$ is the j^{th} task in class C_i.

In other words, each edge connects either the task in C_i to a task in C_{i+1} or a task in C_N to a task in C_1. The processing performed by each of the tasks in C_i is captured by function $f_i : M^+ \rightarrow M^-$, which consumes a set of incoming messages and produces a set of outgoing messages.

4.1.2 GDS Mathematical Formalization

In this section, we formally define the CPC formalization for Graph Dynamical Systems (GDS). The GDS describes a system and its dynamics. In GDS the structure of the graph (connecting edges) may change from iteration to iteration. The change happens as a result of processing in the previous iteration. Therefore, the message in addition to the source and destination carries a time-stamp as well.

Using the mathemtical formulation from Section 4.1.1, all edges $e = (u_{i,j,\tau}, v_{k,l,\tau}) \in E$ obey the constraint

$$k = \begin{cases} i+1 & \text{if } i < N; \\ 1 & \text{if } i = N. \end{cases}$$

In other words, each edge connects either task in C_i to a task in C_{i+1}, or a task in C_N to a task in C_1, and carries a time stamp (τ).

4.1.3 The Constrained Producer-Consumer Algorithm

In this section, we show a generalized form of the CPC parallel-algorithm. The entities in this model can be characterized by a multi-partite interaction graph as shown in Figure 5.2 on page 13. The vertices represent the tasks, and the directed edges correspond to the communication between tasks.

Based on the type of computation they perform, the tasks are divided into N distinct sets of nodes. Further, the tasks in successive classes act as producers and consumers. In each step, the consumers ensure that they have received all the messages sent to them before they begin to process them.

A brief description of CPC algorithm is given below, and is summarized in Figure 4.2

1: **for** $\tau = 1$ **to** termination condition **do**
2: **for** i = 1 **to** N **do** ▷ Iterate over each class
3: **foreach** $t_{i,j} \in C_i$ **do** ▷ Iterate over each task in the class
4: $M^{-}_{i,j,\tau} \leftarrow f_i\left(M^{+}_{k,j,\tau}\right)$; ▷ process received messages and generate new messages
5: sendMessages($M^{-}_{i,j,\tau}$); ▷ send messages
6: **end for**;
7: $i' \in \begin{cases} i+1 & \text{if } i < N \\ 1, & \text{if i = N} \end{cases}$ ▷ Determine next class
8: **foreach** $t_{i',j} \in C_{i'}$ **do** ▷ Iterate over tasks in next class
9: $M^{+}_{i',j,\tau} \leftarrow$ receiveMessages(); ▷ Tasks in next class receive messages
10: **end for**
11: synchronize(); ▷ ensure all msgs received
12: **end for**
13: **end for**

Figure 4.2 The general CPC parallel-algorithm, where tasks may be on different processors. The termination condition can refer to a specific number of time-steps or a more complex convergence criterion. $M^{+}_{i,j,\tau}$ represents the incoming messages of task $t_{i,j}$ during time-step τ.

The algorithm progresses through several time-steps. A time-step is divided into N phases, one per class. The following are the steps that the algorithm follows:

1. At the start of each phase i, tasks in C_i generate and send messages. The message has a destination task in C_{i+1} if $i < N$. Otherwise, the message is destined for a task in C_1.

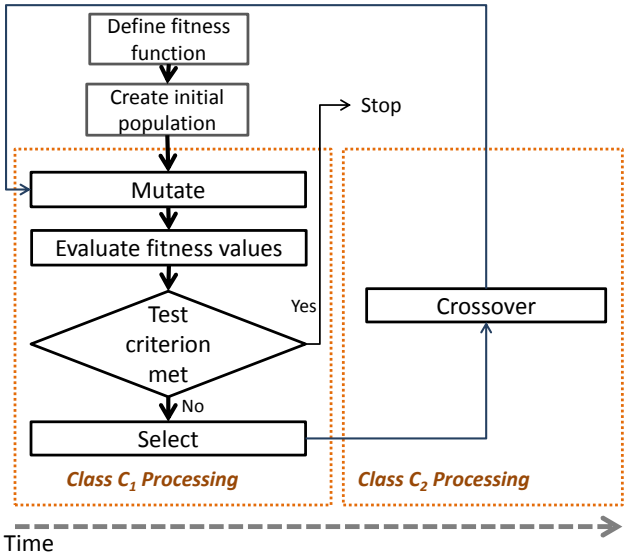
2. After all the tasks in C_i have finished sending their messages, the synchronization happens. This is a global synchronization, which ensures that all the messages have been received by all the consumers.

3. The consumer tasks process their received messages, and update their state before starting the next phase.

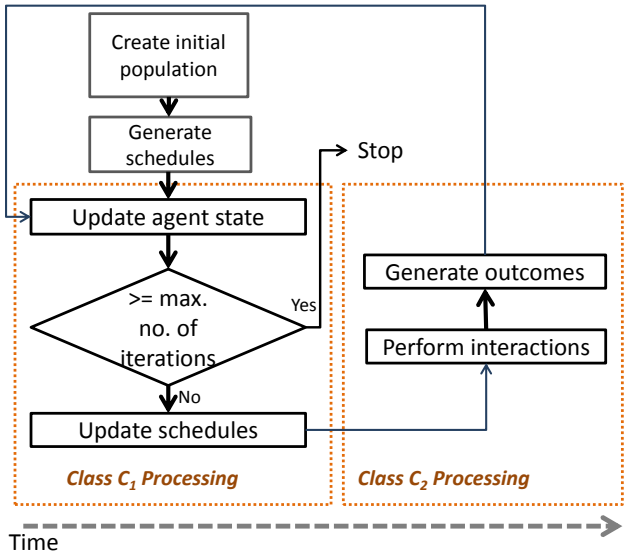
4. When tasks in C_1 receive their messages from the tasks in C_N, the next time-step starts.

4.1.4 Genetic Algorithms

Genetic Algorithms (GA) are well-known for finding solutions to complex problems. Such algorithms are used in engineering to design different kinds products, because they help in finding the right combinations for creating faster and better solutions. They are also used to design computer algorithms, schedule tasks, and solve other optimization problems. GAs work by imitating the way life finds solutions to real-world problems — through the process of evolution. Although GAs are capable of solving complex problems, they are realtively simple to understand.



(a)



(b)

Figure 4.3 (a) Execution of a simple genetic algorithm by two classes of tasks. Tasks in C_1 perform the fitting and selection functions, while tasks in C_2 perform the crossover. (b) A simple agent-based contagion algorithm, showing the interactions between agents at locations. Tasks in Class C_1 perform processing of agents, while tasks in Class C_2 perform processing of locations.

In this section, we discuss Selecto-recombinative genetic algorithms [78,79], one of the simplest forms of GAs, that mainly rely on the use of selection and recombination. We analyze them because they present a minimal set of operators that help us demonstrate the creation of a data-intensive flow. The basic algorithm can be summarized as follows:

1. *Initialization:* Create an initial population. This population is usually generated randomly and can be of any size.
2. *Evaluation:* Calculate a 'fitness' for each individual within the population. The fitness is calculated by how well it fits the desired requirements.
3. *Selection:* Select good solutions by s-wise tournament without replacement [80]. Selection helps us discard bad elements and only keep the best individuals.
4. *Crossover:* Create new individuals by creating a cross between selected individuals, using the uniform crossover technique [81]. This combination attempts to create even 'fitter' offspring for the next population, by inheriting the best traits of both sets of individuals.
5. *Mutation:* Update the fitness value of all offsprings.
6. *Repeat:* Repeat steps 3 — 5 until the desired convergence criteria is met.

The processing of GA on a distributed system happens in two steps as shown in Figure 4.3(a). In the first step, a set of tasks (or mappers) perform the fitting and selection. The selected population is then forwarded to another set of tasks (or reducers). The reducers then perform the crossover, as part of processing in the second step. The outcome is then sent back to the first set of tasks for evaluation and to create a new population. The computation performed by the two sets of tasks is separated in time. The algorithm runs for a number of time-steps, until the criterion is met or the number of iterations are finished. GAs are perfect examples of CPC modeling, where the first set of tasks are mapped to C_1 (to perform fitting and selection), and the second set of tasks are mapped to C_2 (to perform crossover).

4.1.5 Agent-based Contagion Simulations

The agent-based contagion models are based on the reaction-diffusion system across large social contact networks. The network is a bipartite graph of agents and locations as shown in Figure 4.4. Each edge between an agent and a location represents a visit to that location by that agent. The agents produce events and send them to locations to consume. The locations compute interactions between all pairs of co-located and co-spaced visiting agents. If an interaction results in an outcome, the location sends a message to the newly infected agent with information about the outcome. Some of the well-known examples of agent-based contagion models include: containing pandemic influenza at the source [68], spread of infection

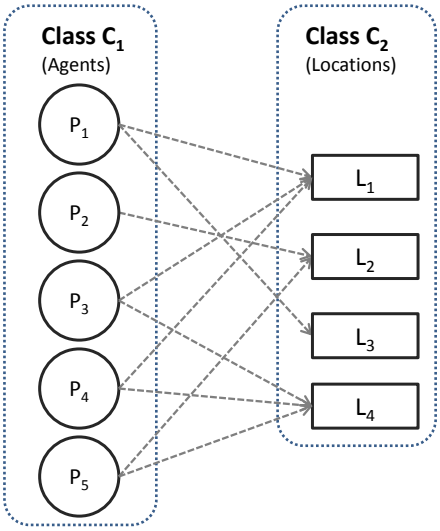


Figure 4.4 A bipartite graph representing the interactions between agents and locations: this refer to Figure 3.1 on page 13. Processing on the graph is performed by two classes of tasks (C_1 and C_2). Agent nodes are assigned to C_1 tasks, and location nodes are assigned to C_2 tasks.

in the population [5, 6, 9, 69], epidemic outbreak [70], spread of disease in human-cells [72, 73], spread of mobile malware [71], coupled modeling of fear and disease [82], and many more. The basic algorithm can be summarized as follows:

1. *Initialization:* Loads or randomly creates the initial population.
2. *Scheduling:* The agents prepare their schedules. For each visit in the schedule list, the agent sends a message to the target location.
3. *Interactions:* The locations compute interactions between each pair of received visits. If an interaction results in an outcome (transmission of disease, information, etc.), the location sends a message to the agent with information of the outcome (infection time, location where the infection occurred, etc.).
4. *Update:* Update the state of agent based on the outcome messages received from locations.
5. *Repeat:* Repeat steps 2 — 4, until the maximum number of iterations (i.e., simulated days) is reached.

Similar to GAs, the agent-based contagion models can be categorized as CPC models, where the tasks in C_1 simulate the agents and the tasks in C_2 simulate the locations as shown in Figure 4.3(b).

In the next section, we perform load quantification for tasks in CPC programs with respect to their incoming and outgoing messages.

4.2 Load Analysis of CPC Algorithms

In this section, we study the computation and communication of CPC parallel programs. We also discuss the relationship between computation and communication. Each task works as a producer and a consumer, alternatively, and is assigned to exactly one class.

An important feature of the tasks in CPC algorithms is the communication and its relationship with computation. The quantitative relationship between the data exchanged (between the producers and consumers), and the amount of computations performed is very application-specific. However, in many cases, the computation performed by a task in a CPC application is related to the number of consumed and produced messages. We further discuss the execution time of a task in relation to the number of messages exchanged (i.e., sent and received by it).

4.2.1 Computations

The processing of a task belonging to class C_i happens in two steps. First, it consumes messages received from a subset of tasks in the producing class (C_j).

$$C_j = \begin{cases} C_N & \text{if } i = 1; \\ C_{i-1}, & \text{otherwise} \end{cases}, C_l = \begin{cases} C_1 & \text{if } i = N; \\ C_{i+1}, & \text{otherwise} \end{cases}$$

This step usually involves updating tables, generating a new population, filtering data elements or a combination of these. Second, it generates messages for tasks in the consuming class (C_l). In this step, the task works as a mapper and performs a selection on the data (i.e., selection, fitting, crossover, classifying, categorizing and mapping, etc.). Therefore, the execution time of a task $t_{i,k} \in C_i$ is a function of the processing performed on messages that are consumed from tasks in C_j, and the computation performed in producing messages for tasks in C_l as shown in Equation 4.1.

$$T \propto g_i(M^+_{i,k}, M^-_{i,k}) \tag{4.1}$$

where $M^+_{i,k}$ is the number of messages consumed by task $t_{i,k}$ and $M^-_{i,k}$ is the number of messages produced by it.

Depending on the application that is simulated, the number of messages produced or consumed by a task may be negligible. For example, in simulations, which model the spread of disease in a population, the number of outcome messages is usually negligible. In this situation, the number of computations performed by the task is a function of the number of messages consumed by it only.

Equation 4.1 shows a general relationship between the number computations performed by a task, and the number of messages exchanged by it. However, the right-hand side function itself is very application-specific, and it estimates the execution time of the task.

As an illustration, in EpiSimdemics [5], where the algorithm models the spread of disease in population, the computation is performed by two classes of tasks. The execution time of a task in the first class is linear to the number of generated messages, and the execution time of a task in the second class is quadratic to the number of received messages. For more details about the quantification of tasks in EpiSimdemics, please refer to Section 4.4.

4.2.2 Communication

The tasks exchange data in the form of messages. If the producing and consuming tasks are located on the same processor, then the message is considered local; otherwise it is remote. As the overhead of the local message is negligible, we only use the number of remote messages (exchanged across all processors) to estimate the communication load. Depending on the application, the communication may or may not be overlapped with execution. The overlap, if it happens, can make the estimation of execution time very difficult. The amount of communication-computation overlap is very application-specific, and a general rule cannot be established to estimate it in advance. In Section 4.4, we use statistical regression modeling to capture this overlap in an example application (EpiSimdemics).

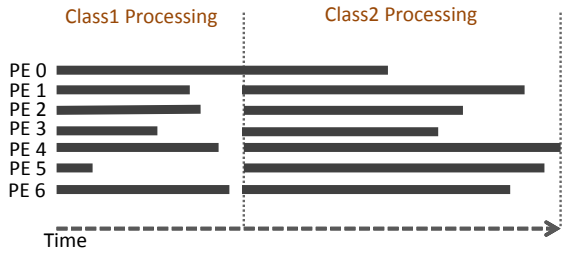


Figure 4.5 The execution times of two classes ($Class_1$ and $Class_2$) of tasks on seven processors. PE 0, when processing $Class_1$ tasks and PE 4, when processing $Class_2$ tasks perform the most computation, and hence keep the other PEs waiting in reaching synchronization.

4.3 The Cost Estimation Metric

In this section, we develop a cost metric that quantitatively measures the cost (execution time) of CPC algorithms. This cost metric, which is an extension of the min-max model

developed in [19], covers the algorithms/models where the computation happens in phases between classes of tasks. The key feature that enabled this approach is the communication load and the synchronization separated computations. Figure 4.5 shows that due to synchronization, the execution cost of a simulation is dominated by the maximally loaded or most imbalanced processors (PE 0 by Class 1 tasks and PE 4 by Class 2 tasks). Therefore, we use the computational imbalance and communication load to estimate the cost of CPC parallel simulations as shown in Equation 4.2.

$$Mcost = k_1 I_1 + k_2 I_2 + ... + k_n I_N + k_c CL \tag{4.2}$$

In the equation, I_i is the computational imbalance in tasks of class i and CL is the communication load. The constants, $k_1, k_2, \ldots, k_n$, and k_c show the relative contribution of components of the cost metric towards the cost of simulation. The metric is applicable to a broad class of CPC algorithms, where the nodes can be represented by a multi-partite interaction graph, shown in Figure 4.1 on page 21.

4.3.1 Cost Metric Components

In this section, we quantify the computational and communication components of the cost metric in terms of incoming and outgoing messages. As shown in Section 4.2, the computational load of a task in a CPC program is a function of the number of messages produced and consumed by it (the degree of the task in the interaction graph). We denote the degree, in degree and out degree of a task $t_{i,j} \in C_i$ by $D^t_{i,j}$, $D^{t+}_{i,j}$, and $D^{t-}_{i,j}$, respectively.

In the distributed processing environment, the tasks are assigned to partitions, where each partition contains one or more tasks of a particular class. Each task is assigned to exactly one partition. The compute load of partition j of class C_i is denoted by $L^P_{i,j}$. It is the sum of the compute loads of all the tasks in that partition and is quantified in Equation 4.3.

$$L^P_{i,j} = \sum_{t_{i,k} \in P_{i,j}} g_k(M^+_{i,k}, M^-_{i,k}) \tag{4.3}$$

where $P_{i,j}$ is partition j of class C_i. The compute load of a class C_i, denoted by L^C_i is the summation of the compute loads of all the tasks in class C_i, and is quantified in Equation 4.4.

$$L^C_i = \sum_{j=1}^{Z} L^P_{i,j} \tag{4.4}$$

where Z is the number of partition of a class. All the classes have an equal number of partitions. The processing of classes is separated in time. Without any loss of generality, we assume that each processor is assigned with exactly one partition. In an ideally load-balanced

assignment of CPC tasks, each partition of a class will have an equal compute load (i.e., equal to the mean compute load of class). The mean compute load of class C_i is denoted by $\overline{L}_i^C$ and is quantified in Equation 4.5.

$$\overline{L}_i^C = \frac{L_i^C}{R} \tag{4.5}$$

A partition is maximally imbalanced in terms of computations of class C_i if it has more computational load than the other partitions of the same class. We measure the computational imbalance in C_i by I_i, which is quantified in Equation 4.6.

$$I_i = \frac{\max(L_{i,j}^P - \overline{L}_i^C)Z}{L_i^C} \tag{4.6}$$

If I_i is zero, then the computation of C_i is perfectly balanced across all the partitions, which means that every partition is assigned tasks (of C_i), such that the total load a partition is equal to $\overline{L}_i^C$. In contrast, I_i is equal to R, when the compute load of C_i is maximally imbalanced (this happens when one partition receives all of the tasks).

After quantifying the computational imbalance, we discuss the communication load. We denote the communication load by CL, which is the ratio of remote messages to total messages. Remote messages are the messages exchanged between tasks assigned to different processors as shown in Equation 4.7.

$$CL = \sum_{i=1}^{N} \sum_{j=1}^{R} \frac{R_{i,j}^p}{|M|} \tag{4.7}$$

where $R_{i,j}^p$ is the number of remote messages sent by partition $P_{i,j}$, N is the total number of classes, and $|M|$ is the total messages, which works as a normalization factor in the equation. The communication load CL is zero when there are no inter-process messages. In contrast, CL is one (maximum) when all the messages are remote.

We use only one component to represent the communication load, and the regression method (that we use in Section 4.3.2 to fit the model and determine their constants) will adjust the communication components and their constants according to the communication latencies in the target application.

After quantifying the computational and communication components of the cost metric, we provide guidelines for extracting their constants. We use the multi-variable regression analysis to fit the model and extract its constants.

4.3.2 Model Fitting and Extraction of Constants

In this section, we outline the methodology for extraction of the cost equation and its constants for any CPC application. For this purpose, We use the multi-variable regression analysis to fit the cost metric and determine its constants. The regression analysis is useful for complex algorithms where a simple analytical solution is difficult to find. The complex systems perform concurrent computation on data elements and produce unpredictable, fine-grain communication requests, which makes it more difficult to apply the standard model reduction techniques such as partial and/or ordinary differential equations [83], approximation by projection [84], and orthogonal decomposition [85].

The regression model expresses the response variable in terms of predictor variables as shown in Equation 4.8 where r is the response variable, $s_1, \ldots, s_n$ are predictor variables, $\beta_1, \ldots, \beta_n$ are regression co-efficients, and β_0 is the constant intercept.

$$r = \sum_{i=1}^{n} (\beta_i s_i) + \beta_0 \tag{4.8}$$

The following are the general guidelines for determining the cost metric and its constants.

1. Identify the cost metric components: The regression method expresses the response parameter (simulation time) as a linear combination of the predictor (independent) variables. Each application will have its own set of independent variables: one computational component for each class and one communication component. In the cost metric in Equation 4.2, $I_1, \ldots, I_N$, and CL are independent variables, while $Mcost$ is the dependent variable.

2. Collect data for all variables: We need to collect data by running the target application on a number of data samples. The data samples are chosen to cover a wide range of input data and to help create a more generalized model. For regression analysis, the required number of samples are at least ten times more than the number of variables [86].

3. Check data for normality: Data collected in the previous step is checked for normality using histogram plotting in a statistical software (i.e., R, SPSS and JMP, etc.). A well-shaped cosine plot shows that the data is normally distributed in ranges for all variables.

4. Perform model fitting: The model is extracted using cross-validation techniques [87]. This requires using two samples of data: $Sample1$ for fitting the model and $Sample2$ for validating the obtained-model. The data collected in step 1 is divided into two samples (about 50% each) using a statistical software. Based on the analysis of correlation and scaler plots of $Sample1$, a multiple regression analysis is employed to fit the model. The model is determined using the residual plot and the R-square value. In the residual plot, if the variances of errors are fairly evenly dispersed around the zero mean line across all the levels of predicted values, the assumption is not violated. This model can be improved by adding linear and non-linear terms (polynomial and exponential, etc.) of independent variables and applying transformations to them. The model fitting can also be automatically performed by using statistical software.

5. Validate the model: The results of the predicted model that are determined using $Sample1$ can be validated on $Sample2$ by comparing the R-square values of both the samples. The R-square value shows the ability of independent variables in explaining the variability of the dependent variable. It varies between 0.0 — 1.0, where a value of 1.0 shows the perfect fit.

6. Check the model for over-fitting: It is very important to check the model for possible over-fitting. The over-fitting can be avoided in three steps. Firstly, validate the model using cross-validation techniques [87]. Secondly, enough samples of data (dependent and independent variables) have to be collected [86]. Finally, over-interpretation must be avoided by means of clearly stating the assumptions that were made (and providing enough details for another researcher to replicate the work). This requires details like the showing the slope of lines do not depend on each other (using multicollinearity VIFs), showing that the dependent and independent variables are related to each other (using scaler plot), stating that the experiments are independent, and performing the normality checks of error distribution.

In the next section, we apply the methodology discussed to determine the cost metric and its constants for EpiSimdemics.

Table 4.1 Coefficiensts for the cost metric components generated using statistical regression modeling.

		Unstd.	Std. Error	Std.	T	Sig.	Stats	
	Constants	Coeff.		Coeff.			Tolerance	VIF
$Constant$		0.003	0.000		27.332	0.000		
I_1	k_1	0.051	0.007	0.061	6.946	0.000	0.994	1.134
I_2_center	k_2	0.747	0.009	0.833	81.855	0.000	0.652	1.534
$I_2\_center\_sq$	k_{2b}	35.813	1.626	0.181	22.023	0.000	0.994	1.006
CL	k_c	0.004	0.000	0.201	19.467	0.000	0.631	1.585

4.4 Cost Model for EpiSimdemics

In this section, we apply regression modeling to determine the cost metric and its constants for EpiSimdemics [5,6,9]. For more information about EpiSimdemics and its implementationi, refer to Section 3.1 and 3.2 of Chapter 3.

EpiSimdemics is a good example of CPC modeling, where person computation is performed by class C_1 tasks and location computation is performed by class C_2 tasks.

4.4.1 Load Analysis

Before applying regression analysis, the user needs to determine a quantification for the computational components in terms of incoming and outgoing messages. Section 3.3 shows that the person computation load can be quantified by the number of messages that it produces ($f_1(t) = M^+_{j,i}$). Similarly, in Section 3.4, we show that the location computation load can be quantified using the number of messages that it consumes.

We use imbalance definitions in Equation 4.6 and load definitions (of persons and locations) in this section to quantify I_1 and I_2 respectively for person and location classes. Here I_1 denotes the compute imbalance in person tasks, and I_2 denotes the compute imbalance in location tasks.

Next, we apply statistical regression analysis to fit the model and determine the constants for the three cost metric components (I_1, I_2, and CL) of EpiSimdemics, where CL represents the communication load of EpiSimdemics.

4.4.2 Data for Regression Analysis

The data for regression analysis was collected from running EpiSimdemics on 732 samples of three data sets: Alabama (AL), Florida (FL) and California (CA). The methodology used to create data for regression analysis is as follows.

We first create 64 partitions of AL, FL, and CA sets using the k-way and multi-constraint features of the Metis [22, 88, 89] partitioning scheme. The Metis-created partitions serve as base cases for generating other samples. The base cases produce 18% remote communication, and an approximate balanced person and location loads (using definitions of person and location loads from Section 4.4.1). We modify the base cases by randomly moving persons and locations from some of the tasks to other tasks to create a total of 732 samples. This creates different person imbalances (I_1), location imbalances (I_2) and communication loads (CL) in each sample. In the 732 samples, remote communication is varied over the range of 18% — 100% of total communication. Because of the nature of the real data, we can not achieve remote communication less than 18%, without increasing computation imbalance. Further, I_1 and I_2 are varied over the range cover of 0% — 100% of mean person load and mean location load respectively, thus forming a representative sample for the space of possible partitions. After collecting the data, we perform statistical modeling in order to determine the cost equation and its constants.

4.4.3 Model Fitting

The statistical analysis was conducted using SPSS [90] version 20.0 on 732 samples with four variables: simulation runtime (T), person computational imbalance (I_1), location com-

Table 4.2 Descriptive Statistics

	Sample1				*Sample2*			
	Min.	Max.	Mean	Std	Min.	Max.	Mean	Std
T	0.008	0.951	0.409	0.209	0.000	1.120	0.390	0.222
I_1	0.065	1.093	0.579	0.255	0.094	1.098	0.585	0.256
I_2	0.039	1.080	0.530	0.238	0.020	1.089	0.517	0.249
CL	0.194	0.978	0.618	0.193	0.195	0.978	0.583	0.196

Table 4.3 Correlations (*Sample1*)

	T	I_2	${I_2}^2$	I_1	CL
T	1	0.943	0.210	0.003	0.660
I_1	0.003	−0.111	−0.049	1	0.213
I_2	0.943	1	0.043	−0.111	0.541
${I_2}^2$	0.210	0.043	1	−0.049	−0.023
CL	0.660	0.541	−0.023	0.213	1

putational imbalance (I_2) and communication load (CL). The histogram plot in Figure 4.6 illustrates that the data is fairly normally distributed for all variables. We divided the 732 samples randomly into two samples (approximately 50% by SPSS). According to the results of the preliminary analysis of $Sample1$, such as descriptive statistics and correlations between variables, a multiple regression analysis was applied to $Sample1$ to fit a parsimonious model that best suits the data. The scatter plot in Figure 4.7 reveals that linearity exists between the dependent and independent variables. The variables I_2 and CL show strong and moderately strong positive linear relationship respectively. Even though I_1 appears to have a weak negative linear relationship with T, however, none of the correlationship between the two independent variables is strong.

As shown in Table 4.1, the final model includes a quadratic term ($I_2\_center\_sq$) for a mean-centered variable (I_2-0.0083) as well as three linear terms (I_2_center, CL, and I_1). The quadratic term was determined while adding the square and cube terms of the independent variables to improve the fitness. The final model was reached when the variances of errors were normally distributed around the zero mean line across all levels in the residual plot shown in Figure 4.8. Independent variables in the final model account for 97.8% of the variability in the dependent variable (r^2=0.956). In Table 4.1, the regression coefficients of I_2_center, $I_2\_center\_sq$, I_1, and CL are positive and statistically significant ($p-value$<0.001), thus implying that an increase in each of the independent variable when controlling for other independent variables also increases the simulation time (T). The results of the final model also depict no presence of the multicollinearity problem, since the degree of multicollinearity (VIFs<2) for each independent variable is very low.

The histogram plot in Figure 4.6 illustrates that the data is normally distributed for all variables. In Figure 4.8 the variances of errors seem to be evenly dispersed around zero across

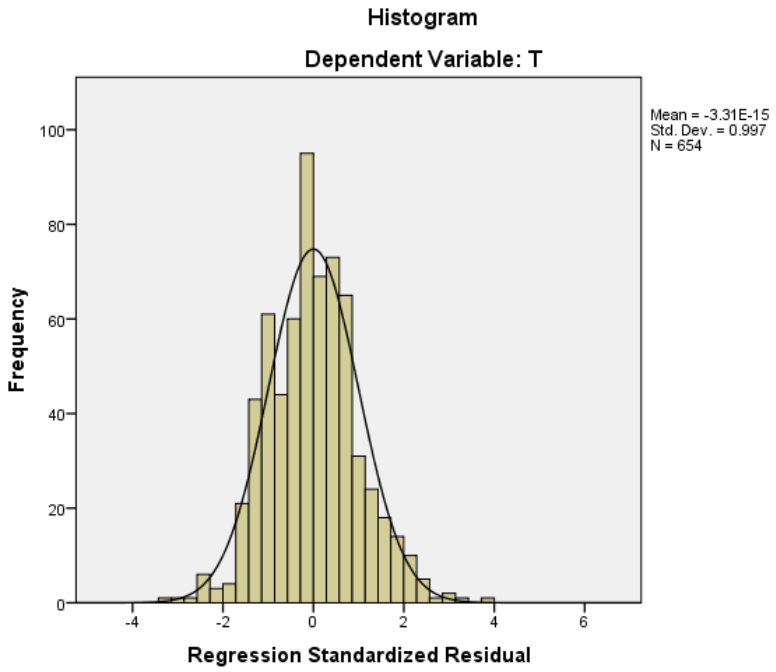


Figure 4.6 The distribution of the residuals are normal for *Sample*1.

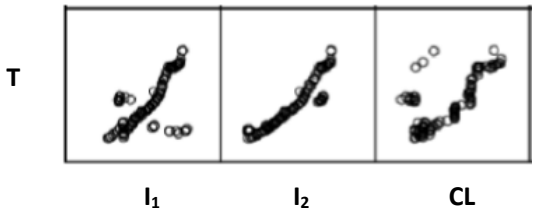


Figure 4.7 I_1, I_2 and CL show linear relationship with simulation runtime.

all levels of predicted values, implying that the equality (or "homogeneity") of variances is not violated.

4.4.4 Model Validation

We evaluate the results of the final predicted model on *Sample*2. To investigate how much variation in T in *Sample*2, is explained by the regression model obtained from *Sample*1, we created a new variable, *predictedT*, using the equation from *Sample*1 and then computed the correlation coefficient ($r = 0.981$). Comparing the R-square (0.962) from *Sample*2 with the R-square (0.954) from *Sample*1, it is slightly larger unexpectedly rather than shrinking.

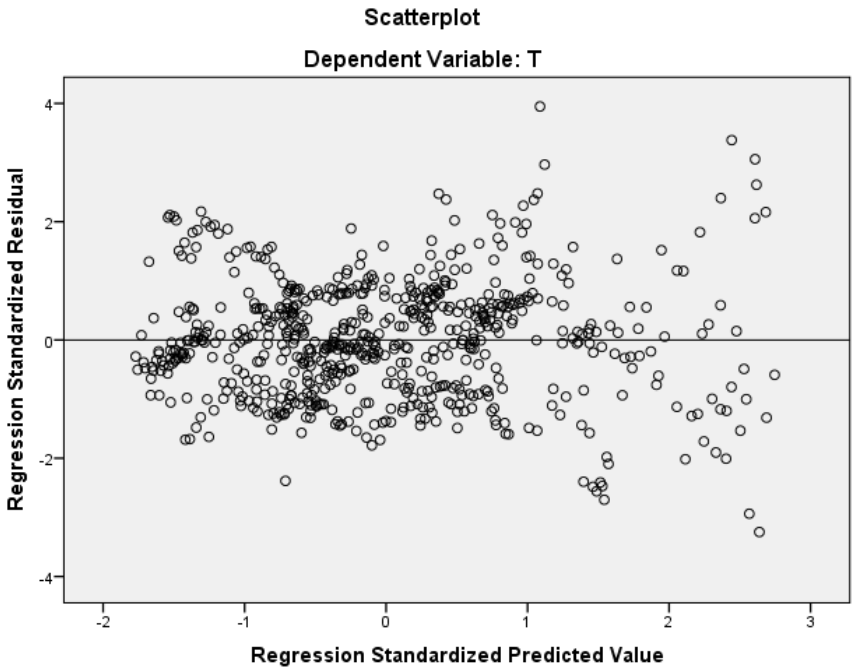


Figure 4.8 Distribution of error do not critically deviate from the homoscedasticity for *Sample*1.

The larger R-square value has no implications, but the applicability of the model to *Sample*2 shows that the model is valid and generally applicable to different data.

We avoided the possible over-fitting by following the three validation steps mentioned in Section 4.3.2.

For the assumption of homoscedasticity, the scatter plot of standardized predicted values and standardized residuals are examined in Figure 4.8. In the plot, the variances of errors seem to be fairly evenly dispersed around zero across all levels of predicted values, meaning that the assumption is not violated.

4.5 Cost Metric Vs Code Profiling

In this section, we discuss the pros and cons of creating a cost metric versus doing a code profiling. The coding profiling is a dynamic analysis technique which helps get the memory usage and CPU utilization during program execution time. It is commonly used by adaptive systems such as just-in-time compilers, dynamic optimizers, power optimizers, and binary translators. Actually, code profiling can be used to develop execution time cost equations. Table 4.4 compares our developed cost metric and code profiling for their pros and cons.

Table 4.4 Cost Metric Vs Code Profiling

	Cost Metric	Code Profiling
1.	Estimates program execution times	Reports actual program execution times
2.	Done using Static Analysis	Done during program runtime
3.	Needs mathematical formulation	Doesn't need mathematical formulation
4.	Needs statistical analysis for extration of constants	Not applicable
5.	Gives time complexity of individual methods	Can not compute time complexity
6.	Do not cover memory usage	Reports memory consumption
7.	Applicable to CPC algorithms only	Applicable to all applications
8.	Relates execution time to produced and consumed messages	Not applicable
9.	Useful in performance analysis	Useful in performance analysis

4.6 Experimental Evaluation

In this section, we show the ability of the cost metric to estimate the simulation runtime. Later in the section, we discuss the ability of the cost metric to determine a sweet-spot for simulation execution time in terms of simulation runtime and number of processors. We also show the ability of the cost metric in highlighting load balancing problems caused by data granularity.

4.6.1 Experimental Setup

Hardware: Our experiments were performed on a high-performance computing cluster consisting of 318 compute nodes. Each node has two octa-core Intel Sandy Bridge CPUs and 64 GB of memory. The cluster uses infiniband (40Gbps) technology for interconnections.

Data: We use EpiSimdemics to simulate the spread of H5N1 avian influenza across social contact networks of populations of the US states. The algorithms presented were evaluated using North Carolina (NC) and California (CA) data sets. Each test is executed for 120 time-steps (i.e., simulated days).

4.6.2 Strong Scaling

In this section, we show the ability of the cost metric in estimating the execution times of EpiSimdemics, when running fixed data sets on an increasing number of processors. We present strong scaling numbers for NC and CA data. The NC data can fit into the memory of a single node, while the CA data requires a minimum of two nodes. The data is partitioned for a one-to-one mapping from partitions to processors. More details about the partitioning schemes (Round-robin and Metis) and their performance in EpiSimdemics can be found in Section 5.3, and Section 5.6 of Chapter 5.

Figure 4.9 compares the cost metric estimated running times and actual running times when simulating NC and CA data. The matching slopes of the actual and predicted curves shows that the cost metric accurately estimates the strong scaling running times for most cases. The lines diverge differently for the two the partitioning methods, thus showing the difference in performance. Further, the cost metric was able to accurately predict this difference. This shows that the cost metric can also be used for performance evaluation of algorithms for a variety of data sets and partitioning schemes. The cost metric is less accurate in predicting the execution times when evaluating a very large number of partitions. One reason for this significant error is the synchronization overheads, as they become dominant once a larger number of processors are in use as shown in Section 5.C of CharmSimdemics [6]. Additionally, the ratio of computation to communication decreases as we distribute data to more processors [91]. The CA, which has a population size that is four times that of NC, shows a smaller error at high partition counts. Figure 4.9(c) shows the error in estimating the execution costs of NC and CA. The error increases up to 6.5% when estimating the execution time for NC on 4K processors. This shows that the cost metric predicts execution times even for a large number of processors with only a small error.

4.6.3 Resource Allocation

Since the cost metric predicts the strong scaling running times with small error, we can use it to perform the cost analysis of studies before launching the jobs. We can determine a sweet-spot in terms of the number of processors, which offers the best performance (time to completion of running jobs in parallel is minimized).

As an example, assuming that one wants to simulate a number of replicas of NC and CA data sets using a total of n processors. One way to do this is to run each set of data individually on all the processors, which would not be efficient if the data sets do not take the same amount of time to execute when running on n processors. Another alternative is to run both the jobs in parallel, each job using half the resources. This is not efficient either; if one data set finishes the task before the other, the job with smaller execution time will finish early, leaving half the resources idle waiting for the compute-heavy job to finish.

A rational solution to do this is to determine a sweet-spot, where the overall time to com-

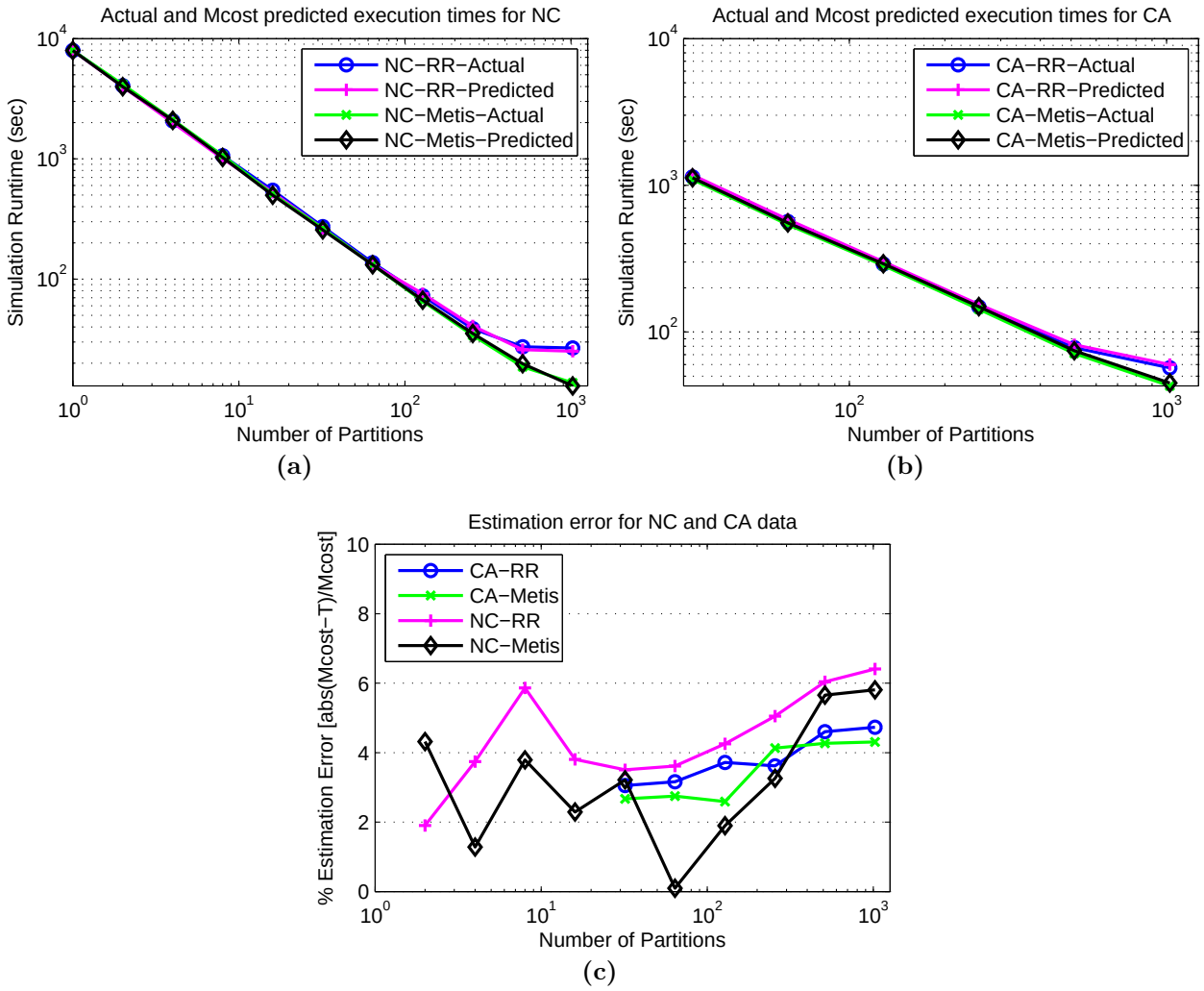


Figure 4.9 Shows the comparison between the cost metric predicted execution times and the actual execution times, when simulating NC and CA data on a number of processors. RR-Predicted and Metis-Predicted refer to the cost metric estimation of simulation execution times, when the data is partitioned using the Round-robin and Metis respectively. RR-Actual and Metis-Actual refer to the actual execution times. (a) The cost metric correctly estimates the running times of NC. (b) The cost metric estimation for CA is better than NC. (c) Represents the predicted and actual execution times. The cost function estimates the execution times with a small error — the largest estimation error being only 6.5% (NC on 4K processors).

pletion is minimized. Using the cost metric, we can determine a sweet-spot in terms of allocation of processors to run each set of data (that minimizes the time to completion). In this particular example, we get the best performance when both the jobs are executing in parallel: NC running on 756 processors and CA running on 3340 processors as shown in the Figure 4.10. This was determined by exhaustively evaluating each n. When running multiple jobs in parallel on a distributed system, the optimal assignment tends to minimize the time

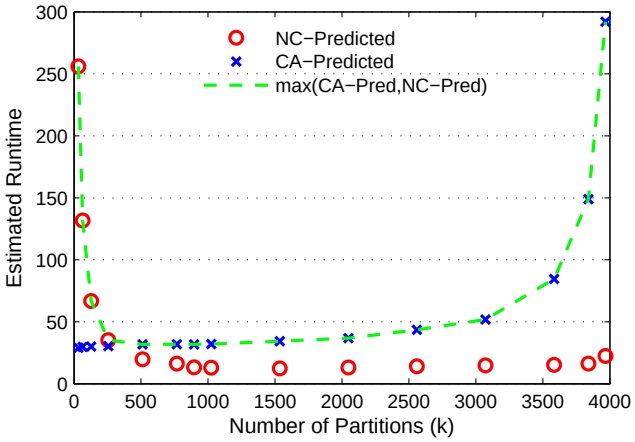


Figure 4.10 Estimated execution times for running NC on k cores and CA on $n-k$ cores. n is the total number of cores (4K in this case). max(CA-Pred,NC-Pred) refers to the time to completion when running both the jobs in parallel on a total of n nodes (NC on k cores and CA on $n-k$ cores). The time to completion is minimized the most when NC is run on 896 cores and CA is run on 3200 cores.

to completion of all tasks.

As an example, we showed the usefulness of the cost metric in resource allocation of NC and CA. However, this metric is general and can be used in job scheduling (resource allocation) of any data. The job scheduling algorithms are well-studied, and we are not claiming any such contribution. Rather our cost metric is used as a cost function to enable the job scheduler to determine the optimal assignment. The metric can work in conjunction with the job scheduling algorithm, where the jobs are assigned an optimal number of nodes, based on evaluation of the cost function. It is not necessary that all the jobs need to run in parallel (all at the same time) to minimize the time to completion. The jobs can run in groups (subset of jobs) and achieve the best performance (minimize the time to completion of running all the groups).

4.6.4 Highlighting Data Granularity Problems

The cost metric also highlighted a problem with load balancing due to the granularity of the location computation of EpiSimdemics. From Section V.B of CharmSimdemics [6], we know that on average the location computation takes about 60% — 70% of the simulation execution time. This shows that imbalance in location computation will affect the total simulation time greatly.

Figure 4.11 shows that the cost metric estimated execution time sharply increases when the number of partitions is increased beyond 512. An increase in the predicted time is an indication of performance degradation, which needs further investigation. Three factors

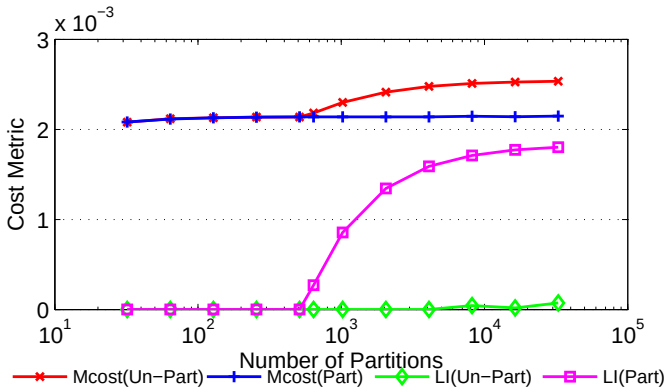


Figure 4.11 Partitioning AL data with Colocation strategy up to 16K cores. Un-part AL contains locations that are large in terms of location computation. Location Imbalance (LI) and $Mcost$ are high when creating a large number of Colocation partitions of AL data in the presence of compute-heavy locations (in terms of location indegrees). After diving compute-heavy locations into multiple smaller locations (part AL data set), the partitions created using Colocation are able to scale up to a larger number of partitions.

contribute to the cost metric estimated time, and Figure 4.11 shows that the location computation is experiencing huge imbalances when divided into more than 512 partitions. A close examination of location data revealed a data granularity problem. Some locations are bigger in terms of location computation load than the ideal partition load (mentioned in Equation 4.5), which makes it impossible for the partitioning algorithms to balance the load across a larger number of partitions.

After dividing the compute-heavy locations into multiple smaller locations, the algorithm can scale up to 16K partitions as shown in Figure 4.11. The splitting was performed in such a way that it does not affect the correctness of simulation results.

The location imbalance with split locations is much smaller compared to the data with compute-heavy locations. The higher scalability of actual strong scaling running times in Figure 5.5(a) on page 54 further confirms this gain, as they were performed with the split locations data.

One question that naturally arises here is with respect to the complete parallelization using location splitting. Dividing compute-heavy locations into multiple smaller locations (Figure 4.11) helps but does not completely solve the problem. At the current finest granularity, the visits to locations are overlapped in time, which makes it extremely hard to achieve a good cut to divide them further. If further divided, the person overlapped in time needs to perform an interaction with all the overlapped persons in that location, resulting in a further increase in the total communication volume.

4.7 Chapter Summary

In this chapter, we developed a cost estimation metric that estimates the execution cost of Constrained Producer-Consumer algorithms. We scientifically extracted and validated the metric using statistical regression analysis and extracted its constants for an example application, EpiSimdemics. The metric predicts the execution times with a small error for a variety of data sets. The metric was also useful in the optimal allocation of processors for the simulation of different data.

Chapter 5

Static Load Distribution

In this chapter, we develop load distribution strategies to improve the efficiency of agent-based epidemic simulations. We start with a discussion of computation and communication workloads in EpiSimdemics, and then, discuss the different load distribution schemes that we have developed to improve its scalability on high-performance clusters.

5.1 Static Load Quantification of EpiSimdemics

We use the load definitions from Chapter 3 to assign weights to the person and location entities of EpiSimdemics. According to these definitions, the persons perform a similar number of computations, and are assigned a weight of 1. The location computation is quantified in terms of its incoming messages. Accordingly, the person computation load in a partition/processor is measured in terms of the number of persons assigned to it, and the location computational load in a partition is measured as the summation of the incoming edges. On the other hand, the communication load is measured by using inter-process messages. Given these load definitions, we discuss the Round-robin, Colocation, Metis and MetColoc data distribution schemes, along with their strengths and weaknesses for partitioning the EpiSimdemics input bipartite graph.

We compare the performance of load distribution strategies to Round-robin scheme. **Round Robin (RR)** is the most basic and simple approach for distribution of data objects to processors. It has been used across applications for comparison against other load distribution schemes [92, 93]. This strategy does not consider the relative weights (compute loads) of persons and locations and simply assigns them to partitions in a round-robin fashion. Round Robin distribution has no overhead and can be easily conducted at the start of simulation. The distribution it creates is usually imbalanced in terms of location computation and communication. However, it provides balanced workloads in terms of person computation (because the persons carry a similar computational load).

Table 5.1 Description and performance comparison of the Static Data Distribution Strategies for North Carolina (NC) Data.

Scheme	Description	Input Graph	Remote Comm.[1]	Person Workload	Location Workload	Improvement (comp. to RR)
Round-robin	Equal number of persons and locations to each processor	V:10.8M E:47M	99%	Bal.	Imbal.	1X
Colocation	Equal number of persons Equal number of location in-edges Co-locate persons and home locations	V:10.8M E:47M	50%	Bal.	Bal.	2.9X
Metis	Partition person-location weighted graph	V:10.8M E:47M	5%	Tuned	Tuned	3.2X
MetColoc	Partition person-location weighted merged graph	V:2.3M E:23M	5%	Tuned	Tuned	3.2X

[1] when creating 16 or more partitions

5.2 Colocation (C)

Colocation (C) is a custom scheme that performs the distribution of locations and persons to processors with a simple and a lower-overhead greedy approach. It balances the computation by attempting to assign the same number of persons and the same total location indegree to each partition. The Colocation scheme uses person-home-location relationship to achieve locality. On a minimum, 46% of the time, the persons visit their home locations, and to exploit this locality, the algorithm assigns persons and their home locations to the same partition.

The algorithm assigns home locations, non-home locations and persons to partitions in a three-step process.

1. This first step assigns home locations to partitions. We first determine the number of persons belonging to each of the home location. Each person is associated with one home location, while a home location may have multiple persons associated with it.

Then, we sort these home locations based on the number of households (the number of persons associated with one home location) in a descending order. Finally, we assign the sorted list of home locations to partitions (one partition at a time) starting with the largest, in such a way that each partition is assigned home locations, where the total number of its households is equal to the mean person load. The mean person load is the total number of persons averaged over the number of partitions. The distribution may not result in a perfectly balanced load as some of the partitions may receive more than the mean number of persons per partition. However, the differences in loads across partitions would be very small, as the number of households per home locations is also very small (1-10), which may result in a difference of no more than 1% of the mean person load.

2. The second step assigns the non-home locations to partitions, based on location indegree (incoming edges). The non-home locations are first sorted based on the indegree. Then, starting with the largest non-home location, we assign locations to partitions (one at a time). The goal is to build partitions such that the sum of location indegree (of the home and non-home locations) across all the partitions is equal to the target partition size. The target partition size is equal to the mean of location indegree (sum of all location indegree averaged over the number of partitions). Again, this distribution may not result in a perfect load distribution, as some of the partitions may receive more than the mean location indegree. The amount of this imbalance depends on the granularity of the location indegree and the number of partitions created. Section 4.6.4 discusses this effect, and outlines a vertex splitting mechanism to overcome this.

3. The third step distributes persons to partitions. We co-locate persons with their home locations (person is assigned to the same partition where his home location is assigned). The home locations are assigned in step 1. This way, each partition gets roughly an equal number of persons. This distribution step can also be merged with step 1.

5.3 Metis (M)

Metis [21, 22, 88] is a widely used library for partitioning graphs and meshes. The algorithms are based on multi-level recursive-bisection, multi-level k-way, and multi-constraint partitioning schemes. The partitions created by Metis (M) reduce the remote communication and balance the computation.

Our input to Metis is a weighted person-location bipartite graph as shown in Figure 5.1(a). Person and location vertices are weighted and so are the edges connecting them. Edge weights are the number of messages sent from person tasks to location tasks on that link in a single time-step. In EpiSimdemics, the edge weights of the person-location bi-partite graph refer to the number of messages sent from a person to a location on that link in one iteration day. The person vertex is assigned a weight of 1, and locations are weighted by thier indegree.

Since person and location processing is separated in time, we assign two weights to each one of them. For persons, the second weight is zero, which means that the person processing happens only in the first phase of the time-step: during the second phase, the persons do not perform any computation. Similarly, the locations are assigned only a second weight (the first weight is zero), which means that location processing happens in the second phase. This is a hint to the Metis partitioner to treat the person and location as different classes of vertices and balance both individually across all partitions.

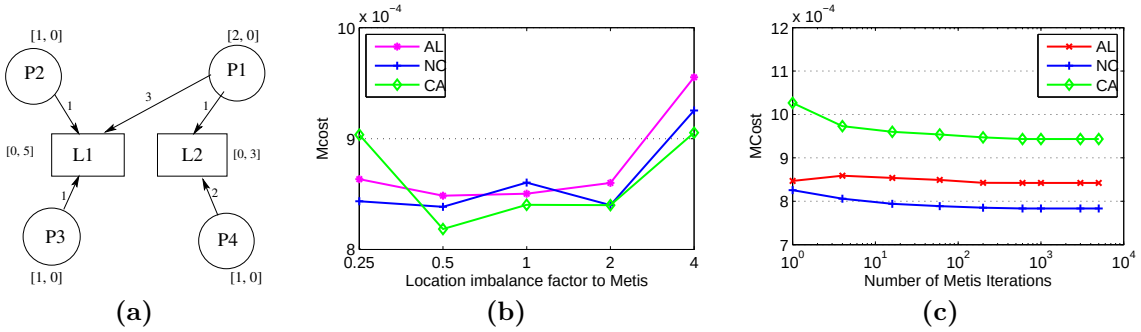


Figure 5.1 (a) Metis input graph: Each node has two weights associated with it. First is the person weight: if the node is a person node, it is 1, otherwise it is 0. Second is the location weight (weight of incoming edges): if the node is a location node, it is non-zero, otherwise it is 0. Edge weight is the number of messages exchanged through that link in a single time-step. (b) Metis gives best partitioning when the Metis imbalance constraints are set at 0.5% and 1.25% for location and person respectively. (c) Shows the optimal number of Metis iterations for AL, NC, and CA data sets. CA is our largest dataset, and puts an upper-bound on the number of Metis iterations required for all other datasets.

Before applying the Metis partitioning algorithm in order to partition the person-location weighted graph, we need to find the imbalance factors for person and location vertices. We investigate these factors for persons and locations in relation to the ratio of the two computations in EpiSimdemics as shown in Figure 3.4 on page 16. The location and Person computations take 65% and 25% of EpiSimdemics runtime respectively. Therefore, we experiment with the imbalance ratios of 1 — 2.5 for location and person respectively.

To attain the optimum imbalance factors (a requirement of Metis partitioner) for person imbalance and location imbalance, we partition the AL, NC, and CA datasets with Metis, using different combinations of person and location imbalances. Figure 5.1(b) shows that Metis gives best-quality partitions (by optimizing the cost metric the most), when person and location imbalances are set at 1.25% and 0.65% respectively. For more details about the imbalances and how they are computed for person and location nodes in EpiSimdemics, please refer to Section 4.3 and 4.4.

We also need to find the optimal number of Metis iterations that produce best-quality partitions. For this purpose, we run Metis with a number of iterations ranging from 1 to 5000 for AL, NC and CA data sets. Figure 5.1(c) shows that the cost metric decreases up to 1000 iterations. However, the increasing number of Metis iterations beyond 1000 does not

lower the cost metric estimated compute times any further. Our largest data (CA) gives an upper bound on the number of iterations that are required for other datasets.

We use the Metis direct k-way and multi-constraint partitioning to partition the person-location bi-partite graph as shown in Figure 5.1(a). We prefer multi-level k-way partitioning over multi-level recursive bisection, as it is faster by a factor of $O(logk)$, and produces better quality partitions [23]. Moreover, the produced partitions have fewer connected edges between them. Further, the minimization of resulting edges helps in achieving higher task locality (local communication).

Additionally, to utilize the benefits of parallel partitioning, we use ParMetis [94], parallel Metis partitioner, to partition our person-location weighted graph.

5.4 MetColoc (MC)

MetColoc (MC) is a custom partitioning strategy that augments Metis by adding knowledge about the person-home-location relationship. The algorithm reduces the input graph to partition, by way of merging person nodes and their respective home-location nodes into super nodes. The merged graph is then partitioned using ParMetis. In terms of its performance, the partitioning quality is similar to Metis (which partition the original graph). However, due to processing on a reduced graph, it has lower-overhead in terms of partitioning runtime and memory consumption.

Table 5.2 Number of vertices and edges in Metis and MetColoc input graphs for the AL, NC and CA data.

	Metis		MetColoc		Reduction in Graph Size (Compared to Metis Graph)	
	Vetices	Edges	Vertices	Edges	Vertices	Edges
AL	5,572,150	16,941,825	1,198,945	11,810,935	21%	69%
NC	10,830,731	33,264,830	2,289,167	23,472,046	21%	70%
CA	40,766,950	91,929,138	7,178,611	63,571,666	17%	69%

Figure 5.2(a) shows a weighted person-location bi-partite graph. We reduce the graph by merging person nodes and their respective home location nodes into super nodes. The merged graph has a reduced number of nodes and connecting edges as shown in Table 5.2 (at least 78% reduction in the number of vertices and 30% reduction in the number of edges). In Figure 5.2(a), HL1 is the home location of P1, P3, and P4 while HL2 is the home-location of P2 and P5. Figure 5.2(b) shows that HL1, P1, P3 and P4 can be merged into one super node (S1), while HL2, P2 and P5 can be merged into another super node (S2). Figure 5.2(c) shows the weighted merged graph with super nodes (S1 and S2) and non home-location nodes (L1 and L2). The non home-location nodes will keep their weights from Figure 5.2(b). The super nodes, and their edges get the merged weights. More specifically, the weights of the

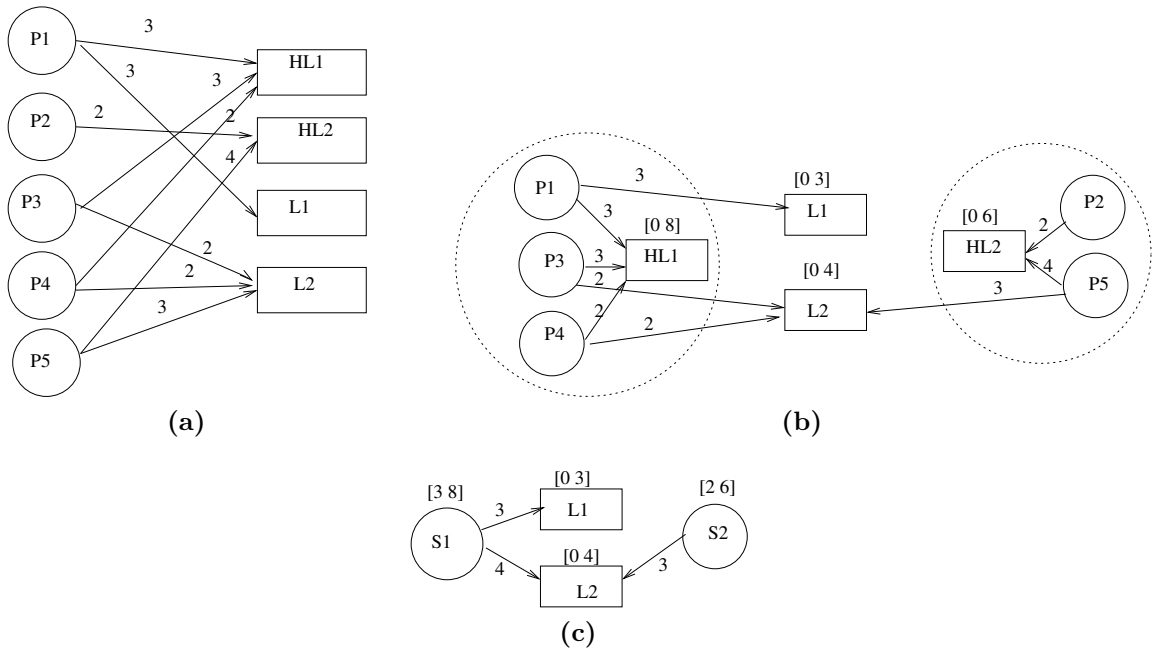


Figure 5.2 (a) Input bipartite graph with person (P1,...,P5), non-home location (L1 and L2), and home location (HL1 and HL2) nodes. (b) Person nodes P1, P3, P4, and their home location node HL1 can be merged into a super-node. Similarly, the person nodes, P2, P5 and their home location node, HL2, can be merged into a super node. (c) After merging, the number of vertices and their connecting edges are reduced significantly.

super nodes are the summation of weights of their original person and home location nodes. Similarly, the weights of their edges is the summation of weights of their person-home-location connecting edges.

After developing the merged graph, we partition it using the Metis partitioning scheme discussed in Section 5.3.

5.5 Performance Evaluation

This section evaluates the Round-robin, Colocation, Metis and MetColoc strategies in terms of performance and partitioning overheads. To help compare the performance of load distribution strategies for a range of partitions and datasets, we partition the input bi-partite graphs of AL, MA, and NC into 4, 16, 32, 128, 512, and 2048 partitions. The experimental data is not the one used in extracting the cost estimation metric in Chapter 4.

5.5.1 Cost Metric based Performance

The performance numbers in this section use the computation and communication load definitions from Chapter 4. Figure 5.3(a) shows that Metis and MetColoc exploit locality more than the Round-robin and Colocation schemes. When creating four or fewer partitions, for all three datasets, only 1% of communication is remote. But, this number increases as we create more partitions, and the remote communication volume increases up to 47% when creating 2048 partitions. However, it is still much smaller in comparison to the Round-robin scheme, where more than 95% of communication is remote (when creating 4 or more partitions). In comparison, the Colocation scheme performs better by exploiting person-home-location locality. Further, for any data, and any number of partitions, at most 54% of communication remains remote.

As persons carry similar computational weights, the Round-robin and Colocation schemes provide well-balanced person workloads as shown in Figure 5.3(b). Metis and MetColoc experiences little person imbalance, but it is never more than 1.25% of the mean person load.

Figure 5.3(c) compares the performance of load distribution schemes during the assigning of locations to partitions. On the one hand, it shows that the Colocation scheme performs best when balancing the location computation, even in the case of creating a large number of partitions. On the other hand, Metis and MetColoc perform similarly and produce close-to-balanced location loads. Locations carry variable computational loads, and as expected, Round-robin fails to achieve a balanced location computation.

When comparing the cost metric estimated costs in Figure 5.3(d), the Round-robin scheme shows higher values, implying its inability to perform a good partitioning. This is because this scheme could not balance location computation and was not able to exploit locality. Metis and MetColoc perform best in the case of optimization of the cost metric. Moreover, in all the cases, its performance (minimization of cost metric numbers) is at least two times better than the Round Robin scheme.

Metis and MetColoc exploit locality, while balancing person and location computations at the same time. In contrast, the Colocation scheme does not exploit locality as much as Metis and MetColoc do, but perfectly balances person and location computations. Figure 5.3(d) shows that the Colocation scheme performs better than the Round-robin distribution scheme for any data and any number of partitions. In terms of task locality, the Colocation scheme performs adequately on a smaller number of partitions, and performs relatively well in comparison to Metis and MetColoc on a larger number of partitions.

5.5.2 Partitioning and Runtime Overhead

Figure 5.4(a) compares the partitioning schemes in terms of memory consumption, while creating 64 partitions for the AL, MA, NC, MI and FL datasets. Round-robin does not

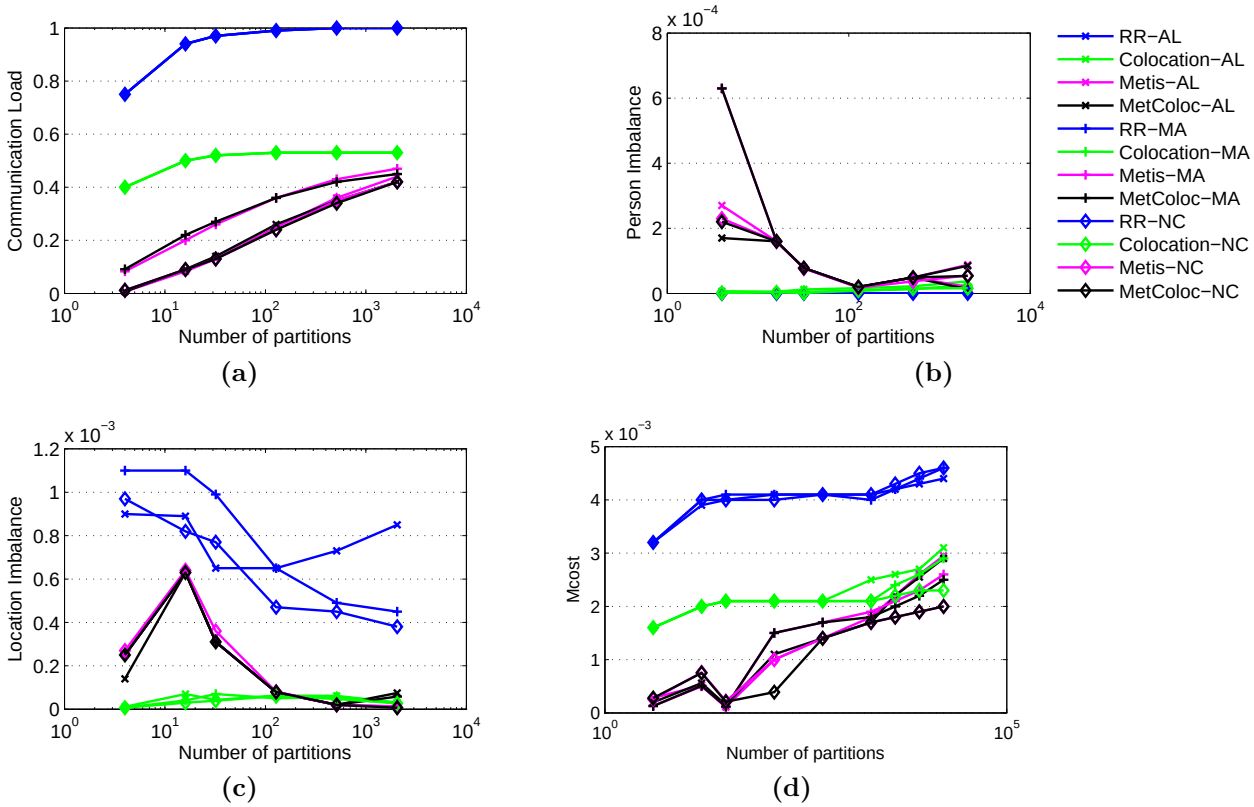


Figure 5.3 Performance analysis of data distribution schemes. (a) The Round-robin distribution scheme provides 95% of remote communication for most of the configurations. When forming more partitions, Metis and MetColoc creates higher remote communication, which catches up with the Colocation scheme on 2K partitions. (b) Round-robin is completely overlapped by the Colocation scheme here, as both give perfect person computational loads. Metis and MetColoc experience a little imbalance (less than 1.25% of the mean person load). c) The Round-robin distribution also performs poorly in balancing the location computation. To reduce remote communication, Metis and MetColoc observe considerable imbalance in location computation on a smaller number of partitions(which can be controlled by setting a smaller location imbalance factor). It gives well-balanced location loads on a larger number of partitions. (d) The cost metric values for Metis and MetColoc increase when the data is divided into a larger number of partitions. This is the effect of an increase in remote communication. Colocation performs similarly to Metis and MetColoc when creating a large number of partitions.

maintain any state during data distribution and has zero memory requirements. The Colocation scheme, however, is a streaming scheme, which maintains very little state, and has the smallest memory consumption. Its memory consumption increases very little when the size of data to partition increases. ParMetis was run on 160 cores, and the memory consumption is the cumulative memory usage across all processors. Figure 5.4 shows that Metis has the largest memory consumption, and it increases linearly as the data size to partition increases. MetColoc, which partitions the merged graph (reduced number of vertices and edges), has half the memory overhead of Metis. However, for both Metis and MetColoc, the memory

consumption increases linearly as the size of the data to partition increases.

We also compare the distribution schemes in terms of partitioning runtime overhead. Again, we run Colocation on one core, and Metis/MetColoc on 120 processor cores. Figure 5.4(b) shows that the Round-robin distribution has zero runtime overhead. Among the other three schemes, Colocation has the smallest runtime overhead. Its runtime overhead is at most 6% of the overhead of Metis, and it stays constant when creating more partitions of the same data. In contrast, the Metis and MetColoc time to partition increases exponentially, as we create a larger number of partitions. MetColoc partitions the merged graph, which takes lesser time in partitioning compared to Metis.

5.5.3 Conclusion

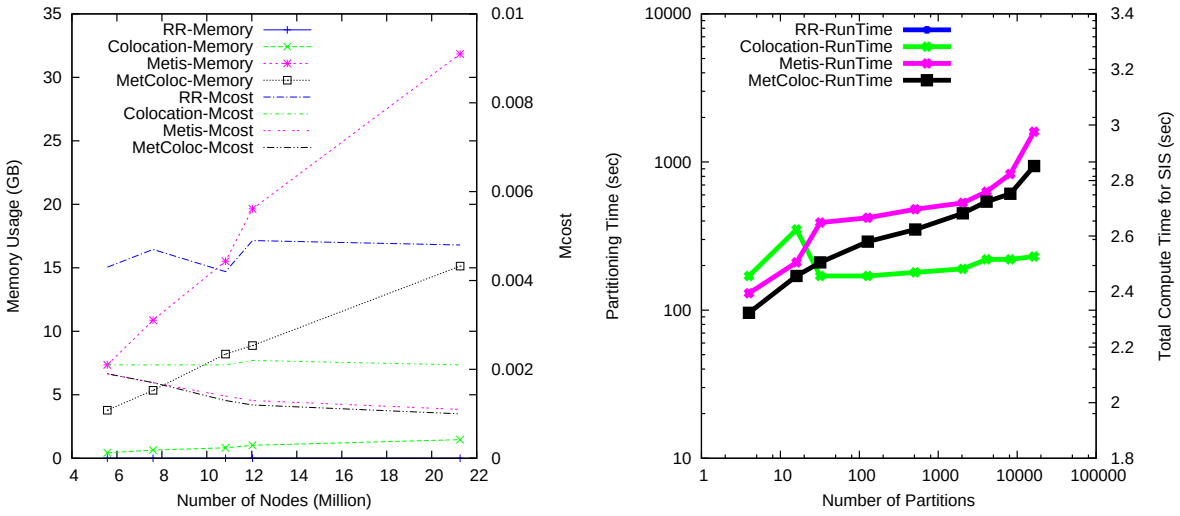
The choice of a partitioning scheme always depends on a tradeoff between performance and partitioning time/memory overhead. Given the strengths and weaknesses of data distribution schemes, it is important to recognize the use of the appropriate scheme(s) for different scenarios. Based on the performance numbers shown in Figure 5.3 and 5.4, we recommend using of Metis/MetColoc when creating a smaller number of partitions of any data, and using the Colocation scheme when creating a larger number of partitions of any data. The choice can also be limited by the size of memory of the system used for partitioning as shown in Figure 5.4. For example, FL data, (21 million nodes) cannot be partitioned by using the Metis scheme on computing machines that have a memory of 21 GB or less, but it can be partitioned using the other three schemes.

5.6 Experimental Performance Evaluation

In this section, we evaluate the impact of the Round-robin, Colocation, Metis and MetColoc strategies on the strong scaling performance of EpiSimdemics. We also show the ability of the cost metric in estimating the simulation runtime.

5.6.1 Experimental Setup

Hardware: Our experiments were performed on a high-performance computing cluster consisting of 318 compute nodes, where each node had two octa-core Intel Sandy Bridge CPUs (model E5-2670) and a memory of 64 GB. The cluster used infiniband (40G) technology for interconnections.



(a) AL, GA, NC, IL, and TX data

(b) NC data

Figure 5.4 (a) Memory consumption for Metis and MetColoc increases linearly as the size of the graph to partition increases. b) Although the Colocation scheme and Metis/MetColoc show a similar performance on a larger number of partitions, the partitioning runtime overhead of Metis and MetColoc increases exponentially when creating a larger number of partitions. ParMetis was run on 160 cores.

Software: We used Charm++ v6.4 [58] for our experiments, having enabled the message buffering scheme. The buffering scheme helped in reducing communication traffic by bundling messages at the source and intermediate destinations.

Data: We used EpiSimdemics to simulate the spread of H5N1 avian influenza across social contact networks of the US states. The algorithms presented were, thus, evaluated using AL, NC, and CA datasets. Each test was executed for 120 time-steps. For all our experiments, the attack rate (the ratio of the number of infected people to the total number of people) is roughly 40%.

5.6.2 Strong Scaling

We first present the performance numbers for fixed problem sizes of Alabama (AL), North Carolina (NC), and California (CA). The chosen datasets cover a wide range of our input data. The AL and NC can fit on a single node, while the four times larger dataset, CA, can fit on a minimum of two nodes. Therefore, we use running times of EpiSimdemics on one node (one core) and two nodes (two cores) as a base case for calculating the speedup of AL-NC and CA respectively. Data, pre-processed for static load balancing, is assigned to processors in a

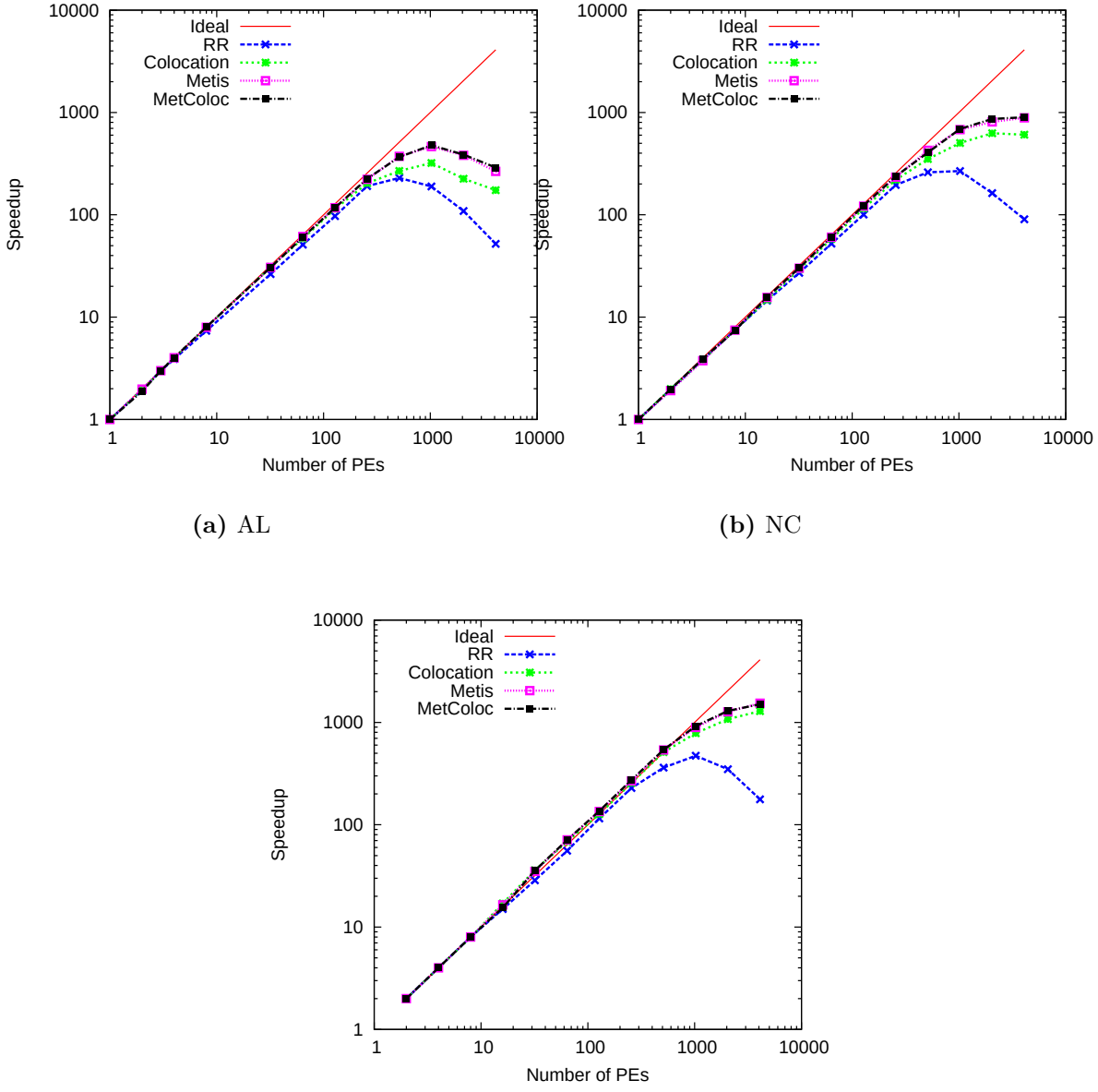
one-to-one mapping of partitions to processors. The strong scaling evaluation includes the discussion of speedup, actual running times and the cost metric estimated execution times.

Figure 5.5 shows the speedup numbers of the four distribution schemes for all three datasets. The ideal line marks the perfect speedup and is used as a reference point. The Round-round based distribution fails to scale for all three datasets. This is primarily due to the increased remote communication and imbalanced location computation. The Round-robin distribution performs quite poorly in exploiting the task locality, and even during creating a smaller number of partitions (16), more than 95% of communication is remote as shown in Section 5.5. Locations carry variable computations, while the Round-robin distribution of locations to partitions creates imbalanced location computational workloads. Metis and MetColoc are, hence, the most scalable in all cases. This is mainly due to their ability to balance person and location computational loads, while reducing remote communication even on a larger number of partitions. The Colocation created partitions that scales better compared to Round-robin for all data. In comparison to Metis/MetColoc, Colocation performs similar, while creating a larger number of partitions. The Colocation scheme achieves this by perfectly balancing computational loads and by exploiting person-home-location locality.

In summary, all the schemes scale better compared to Round-robin. However, for AL and NC data, using more PEs beyond the scaling peaks, does not achieve a better performance. This is because the amount of computations in comparison to communication decreases as we distribute data to more processors. The data is split among more and more processors, and it comes to a point where each processor will not have a sufficient amount of computations to amortize the costs of communication and synchronization [91]. The CA, which is four times larger than NC, has shown better scaling as shown in Figure 5.5(c).

5.7 Chapter Summary

In this chapter, we adopted the Metis graph partitioning library for partitioning the EpiSimdemics input bi-partite graph. We also developed two custom schemes, the Colocation and MetColoc, to perform static load distribution in EpiSimdemics. The semantic-aware schemes improved the strong scaling performance by 3X in comparison to the Round-robin. Furthermore, Metis showed best performance, but at the cost of higher memory and partitioning overhead. MetColoc showed similar performance as Metis, but with half the overhead of Metis. Finally, the Colocation performance appeared to be a little less compared to Metis and MetColoc, but had much smaller partitioning and runtime overhead, especially for a larger data and a larger partitions count.



(a) AL

(b) NC

(c) CA

Figure 5.5 Actual strong scaling speedup of partitioning strategies for AL, NC, and CA data sets. (a) With AL, due to a small amount of computations on a large number of cores, all the partitioning schemes show a moderate performance. However, the semantic-aware schemes still perform better than Round-robin. (b) With NC data, the partitioning schemes scale better compared to AL. On 2K PEs, Metis, MetColoc and Colocation achieve an efficiency of 30%, 39%, and 40% respectively. The Round-robin performs the worst and achieves 6% of efficiency. (c) With our largest data as CA, the partitioning strategies scale even further, and on 4K cores, they achieve an efficiency of 31%, 37%, and 36% for the Colocation, Metis, MetColoc schemes respectively. On the same number of cores, Round-robin is only 8% efficient.

Chapter 6

Disease Dynamics and Computational Load

In this chapter, we study the effects of the disease model parameters on the compute time in EpiSimdemics. More specifically, we study the individual and joint effects of transmissibility, infectious period and incubation period on the execution time in the SIR, SIS, and SI disease models. We also develop analytical equations to quantify these effects.

6.1 Disease Models and Parameters

6.1.1 Disease Model Parameters

In this section, we discuss the disease model parameters of a networked Suscetible-Exposed-Infectious-Recovered (SEIR) disease model. The disease states of the SEIR model are used to describe the progression of a disease in an individual and its transmission in the population [7] as shown in Figure 6.1. To simplify the disease process, we discuss the three most important parameters: transmissibility, the infectious period and the incubation period. The transmissibility (ρ) controls the transmission intensity of a disease in the population. It is the probability of transmission of a disease from an infectious individual to a susceptible individual in one minute of contact. The incubation period (Δt_E) is the interval during which an infected individual cannot transmit the disease to other susceptible individuals. The infectious period (Δt_I) is the period during which an infected individual can transmit the disease to the susceptible individuals. Table 6.1 lists the dwell times in different disease states for typical influenza.

Table 6.1 Typical Influenza ranges for the disease state parameters.

Disease State	Math Term	Typical Influenza Range (Days)
Exposed	Δt_E	1 – 2
Infectious	Δt_I	3 – 7
Recovered	Δt_R	∞[1]

[1] fixed at simulation length

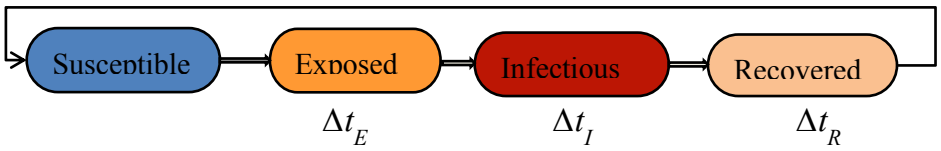


Figure 6.1 The four states of a basic Susceptible-Exposed-Infectious-Recovered (SEIR) disease model. Δt_E, Δt_I, and Δt_R show the dwell times in exposed, infectious, and recovered states respectively. The model can be used to describe the SIR, SIS and SI disease models.

6.1.2 Disease Models

Based on the dwell times in the exposed, infectious, and recovered states, the diseases can be modeled using the SIR, SIS, and SI disease models as shown in Table 6.2. In SIR (Susceptible→Exposed→Infectious→Recovered), a person even after recovering stays in the recovered state forever, and will not become susceptible again. This individual develops an immunity to that strain of the disease. Typical influenza is usually modeled using SIR [9]. On the other hand, in SIS (Susceptible→Exposed→Infectious→Susceptible), a person after recovering becomes immediately susceptible to the disease again. Measles, a well-known infectious disease, and most other sexually transmitted diseases conform to SIS [95, 96]. In SI (Susceptible→Exposed→Infectious), a person after getting infectious remains infectious forever. The Human Immunodeficiency Virus (HIV/AIDS), usually transmitted through bodily fluids can be modeled using SI [97].

To model the progression of a disease within an agent, we use a probabilistic timed transition system (PTTSs). PTTS is an extension of the finite state machine (FSM) with two additional

Table 6.2 Shows the dwell times in different disease states for SIR, SIS and SI disease models.

Model	Δt_E	Δt_I	Δt_R
SIR	$0 - \infty$	$0 - \infty$	∞
SIS	$0 - \infty$	$0 - \infty$	0
SI	$0 - \infty$	∞[1]	N/A

[1] fixed at simulation length

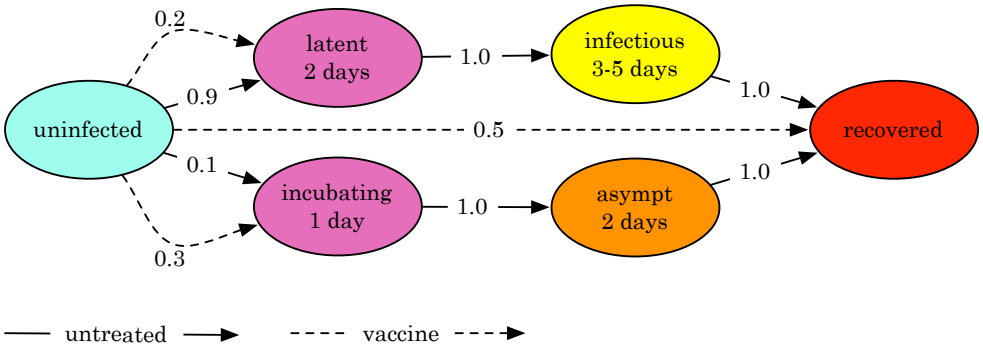


Figure 6.2 A simple disease model that shows the progression of a disease within an agent. Ovals represent the disease states, while the lines represent the transition between those states. The states are labeled and marked with their respective dwell times. The labels on lines further show the probabilities of transitions.

features: the state transitions are probabilistic and timed. PTTS in Figure 6.2 is a simple example of a SEIR disease model, where each state has a name, a dwell time, and a labeled set of weighted transitions to other states. The example shows three paths, starting from uninfected (susceptible) state and ending in the recovered state. The upper path represents the progression of a disease with a two-day latent period, then three to five days of an infectious period, followed by a recovery period. The lower path represents a smaller incubation period of one day, followed by two days of illness. The asymptomatic state (asympt) shows that the individual does not show any signs of disease, but can still infect the others. The dotted lines show the transition, wherein the individual is vaccinated and has some immunity to the disease. The solid lines show the transition (of disease in an individual) from one state to another when the individual is not vaccinated. The labels on the lines are probabilities of transitions.

6.1.3 Number of Interactions

Before discussing the effects of disease model parameters on the execution time, we would like to discuss the number of interactions (major part of location computation), and its relationship with the state of persons at that location. An interaction happens when an infected person comes in contact with a susceptible person. In EpiSimdemics, 60% — 70% of the total simulation time is spent in computing interactions (processing of Equation 3.1) at locations [6]. In Section 3.1, we showed that the number of interactions at a location can be statically quantified by using the number of people that visit that location. However, this quantification may not be accurate, as not all the people who visit a particular location take part in the interactions. Rather, only pairs of people, where one is susceptible and one is infectious, interact. Since people in recovered and incubation state cannot infect others or get infected, they do not interact at locations. During an iteration r (of simulation), the

number of interactions at a location can be computed using Equation 6.1, given that the infectious and susceptible people at that location are overlapped in time.

$$INTERACTIONS_l, r = N_l^2(FS_{l,r} \cdot FI_{l,r}) \tag{6.1}$$

where N_l is the number of visiting people, $FS_{l,r}$ is the fraction of susceptible people, and $FI_{l,r}$ is the fraction of infectious people in iteration r at location l.

We generalize the potential number of interactions across all locations using the fraction of susceptible and fraction of infectious people across all locations. The potential number of interactions across an interaction is approximated using the following equation.

$$PINTERACTIONS_r = N^2(FS_r \cdot FI_r) \tag{6.2}$$

where N is the total number of people, FS_r is the fraction of susceptible people in iteration r, and FI_r is the fraction of infectious people in iteration r. It is a theoritical approximation, and is based on the fraction of infectious and susceptible individuals across all locations.

Since the fraction of infectious and susceptible people may change, the execution time may also differ from iteration to iteration as well. From Equation 6.2, we see that the number of interactions are maximum when an equal number of people are infectious and susceptible ($FS_r = FI_r = 0.5$).

We also discuss the potential number of interactions in different disease models. In SIR, the person after recovering, stays in the recovered state forever, and does not become susceptible again. However, the persons in the recovered state do not take part in any interactions. The number of interactions in SIR decreases as more people enter the recovered state, thus decreasing the fraction of people in infectious and susceptible states as shown in Figure 6.3.

In SI, the person after becoming infectious remains in that state forever. Figure 6.4 shows that the number of interactions in SI are maximum, when half of the people are infectious and the other half is susceptible. However, as the fraction of people in the infectious state increases beyond 0.5 (which means that more than half the population is infectious), the potential number of interactions start to decrease. In SIS, the potential number of interactions remain fixed after the fraction of susceptible and infectious reach a constant point as shown in Figure 6.5. At that point, the number of people in different disease states remain constant. Since the persons will not remain in one state forever, the number of interactions, and, hence, computation time in SIS is usually larger when compared to SIR and SI.

The change in the state of the person is influenced by the disease state parameters (transmissibility, infectious period, and incubation period). The influence is different in different disease models. In the remainder of the chapter, we analyze and quantify the individual and joint effects of these parameters on the execution time in the SIR, SIS and SI disease models.

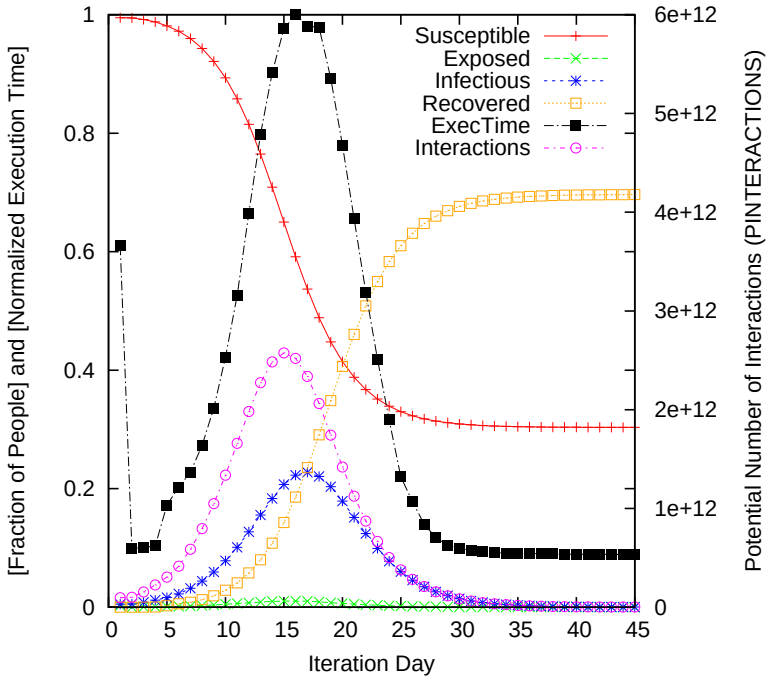


Figure 6.3 Disease evolution using the SIR disease model. The fraction of people in each state (S, E, I, R) on each simulation day is shown, along with the number of potential interactions and normalized compute time (compute time for an iteration is divided by the maximum iteration compute time).

6.2 Experimental Setup

In general, we use the following experimental setup for studying the individual and pairwise effects of disease model parameters on compute time. However, the specific selection of data, disease model and disease model parameters will depend on the type of the study.

Hardware: All our experiments were performed on eight nodes (96 processors) of a high-performance computing cluster consisting of 68 Dell C6220 compute nodes. Each node has a memory of 64 GB. The cluster uses infiniband (40G) technology for interconnections.

Software: We use Charm++ v6.5 for our experiments with message buffering scheme enabled [98]. The buffering scheme helps in reducing communication traffic by bundling messages at source and intermediate destinations.

Data: We use EpiSimdemics to simulate the spread of disease across the social contact network of Alabama (AL) state. Each run is executed for 120 simulation days (time-steps). For all the experiments; five people per thousand (same people every time) were infected at the start of the simulation.

Disease Model Parameters: We perform our experiments in variations of three disease model

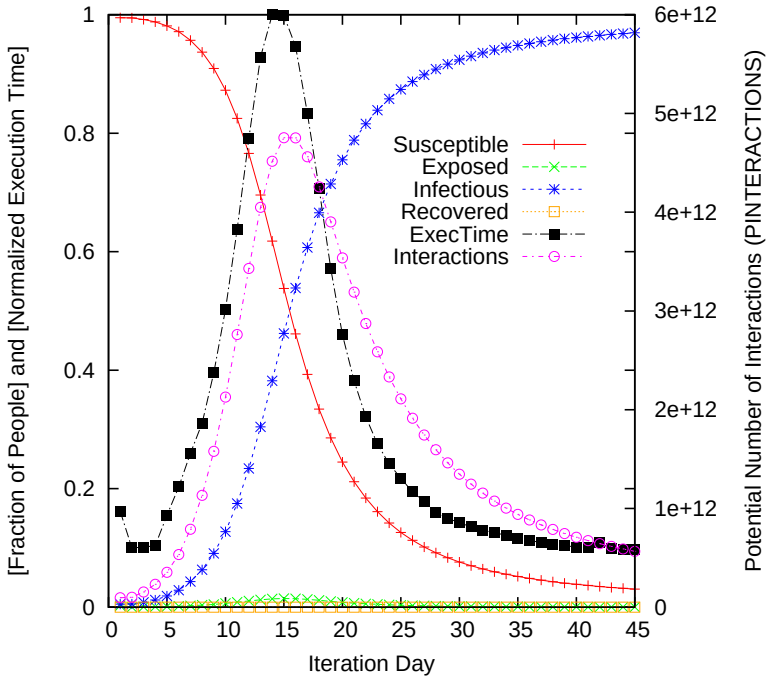


Figure 6.4 Disease evolution using SI disease model. The fraction of people in each state (S, E, I, R) on each simulation day is shown. The number of potential interactions and actual execution time reaches a maximum when approximately half the population is infectious, and half susceptible.

parameters: transmissibility, infectious period, and incubation period. We vary transmissibility in the range of 0 – 1. For infectious period and incubation period, we use 1 – 120 simulation days.

Disease Models: We experiment with three disease models: SIR, SIS, and SI.

6.3 Individual Effects of Disease Model Parameters

In this section, we study the individual effects of the three disease model parameters, namely ransmissibility, infectious period and incubation period on the execution time in SIR, SIS, and SI disease models. In other words, we study the effects of variation in one parameter while keeping the other two parameters fixed. We also extract analytical equations (one for each disease model) that capture the individual effects of these parameters.

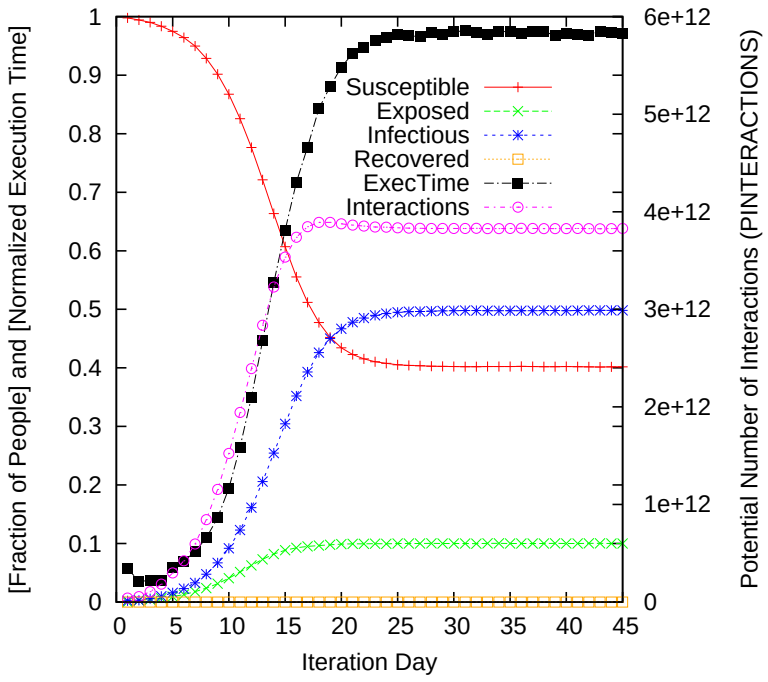
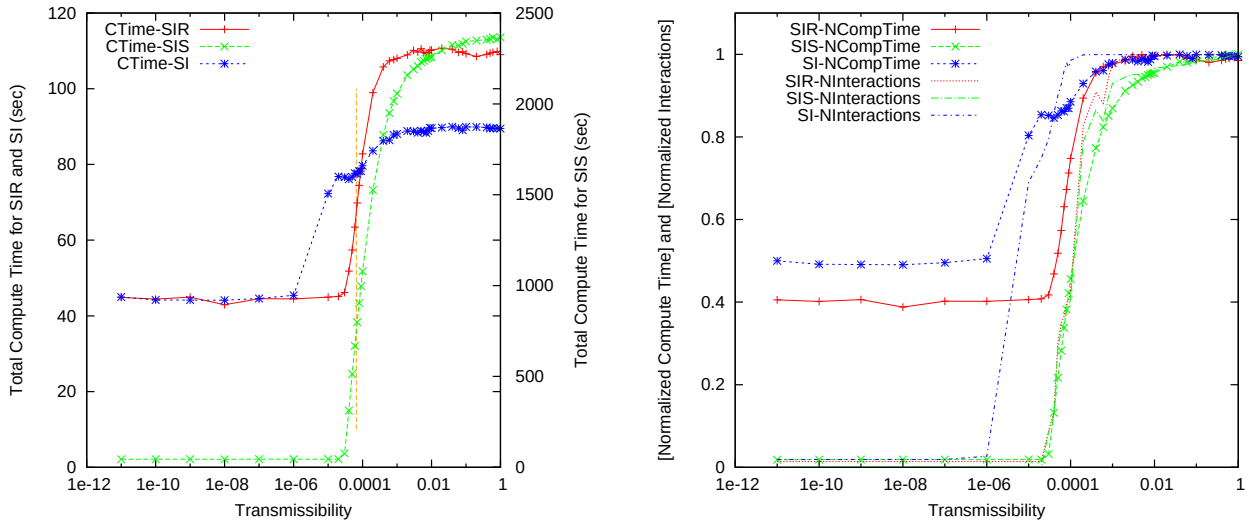


Figure 6.5 Disease evolution using the SIS disease model. The fraction of people in each state (S, E, I, R) on each simulation day is shown. The potential number of interactions and the actual execution times reache a maximum constant after the fraction of people in the infectious and susceptible state stabilize at 0.4 each.

6.3.1 Effects of Disease Transmissibility

6.3.1.1 Analysis

Transmissibility (ρ) is the probability of the transmission of a disease in one minute of contact. In general, increasing transmissibility results in a faster transmission of disease from infectious to susceptible individuals. To analyze the effects of transmissibility on the execution time of different disease models, we use the experimental setup introduced in Section 6.2. In the experiment, we keep the infectious period and incubation period fixed at four iteration days and one iteration day, respectively, and vary transmissibility in the range of 0.0 – 1.0, where 0 implies no transmission of disease for any length of contact, and 1 implies a sure transmission of disease from an infectious individual to a susceptible individual in at least one minute of contact. The infectious period of four days was chosen, because at this number, the compute time is between maximum and minimum (for SIR and SI disease models), as marked by the orange line in Figure 6.8(a). The incubation period of one day was chosen to keep the number of people in this state at minimum. This way, the majority of population stays in susceptible and infectious states, and shows the maximum effect of



(a) **(b)**

Figure 6.6 Effect of an increase in transmissibility on the compute time in SIR, SIS and SI disease models. a) The constant compute time at smaller transmissibility is the time taken by the simulation when no interactions are happening (book keeping, etc.). At a very high transmissibility, the maximum possible interactions have already happened, and the compute time remains constant after that. The vertical orange line shows the points in curves that will be used as constants in the study of individual effects of the infectious period. b) The compute times and number of interactions are normalized by the maximum compute time and the maximum number of interactions in their respective disease models. The shape and the starting point of the normalized interaction curve suggest that the compute time is largely dependent on the number of interactions.

transmissibility on the compute time. We cannot use an incubation period of less than one day, as it is an implementation requirement of EpiSimdemics.

Table 6.3 Epidemic starting points for the three disease model parameters

Model	ρ	Δt_I	Δt_E
SIR	0.00040	3	25
SIS	0.00035	3	42
SI	0.000019	∞[1]	23

[1] fixed at simulation length

Figure 6.6 shows that at very small values of transmissibility, the total compute time (simulation execution time) is small and constant for all disease models. At this transmissibility, the probability of the spread of infection from infectious to susceptible individual is very small. The transmission intensity of a disease in the population is measured using the reproductive number R_0, which is the number of secondary cases for each primary case [99, 100]. At very

small transmissibility $R_0 < 1$, due to which the disease never becomes an epidemic.

Furthermore, in all iterations, the fraction of infectious individuals (FI_r) is smaller than 0.02, which means that less than 2% of the people are infectious at any given time, resulting in a smaller number of interactions and smaller compute times. The absolute compute time is about 49 seconds for all disease models at these lower transmissibility points. This is the simulation execution time when no (or very few) interactions occur.

Beyond, the said transmissibility point, the compute time for all disease models increases exponentially. The transmissibility at which these effects start happening is different for different disease models as listed in Table 6.3. For SI, the compute time starts increasing at a smaller transmissibility compared to others. The reason for this effect is that even on a smaller transmissibility, the fraction of infectious people (FS_r) in SI increases quickly and results in an epidemic. In SIR and SIS, these effects start happening at a slightly larger transmissibility, because the infectious people recover or enter the susceptible state, not leaving enough people in the infectious state to result in an epidemic (at smaller transmissibility).

Regardless of the starting point of the epidemic, the compute time and number of interactions in all three models increases exponentially as the transmissibility increases. However, the increase is more in SIS compared to SIR and SI. In SIS, people do not stay in the recovered state, but become susceptible again immediatly, which means that a larger fraction of people remain in the infectious and susceptible states for a larger number of interations (shown in Figure 6.4). A study performed by by Saha, Adiga and Vullikanti [101] shows that even in SIS, the number of people in the infectious state will ultimately become zero, however, the number of simulations to reach this state are much larger than SIR and SI. That is why we see a higher compute time for SIS simulations.

This results in a larger number of total interactions, and hence larger compute times. Although, this is true for SI as well that the persons never enter the recovered state, but in SI people stay in the infectious state forever, which quickly decreases the fraction of individuals in the susceptible state (FS_r), thus resulting in a lesser number of potential interactions as stated by Equation 6.2 and shown in Figure 6.4.

In Figure 6.6, we also see that an increase in transmissibility after a certain point does not increase the compute time any further. Beyond that point, the compute times remain constant for all disease models. Again, such a transmissibility point is different for different disease models. In SIR, most of the people have already entered the recovered state, and in SI most of the people already entered the infectious state (>99%), thus preventing any further interactions.

In SIS, because a person is susceptible to infection multiple times, the interactions always happen. And this state reaches its maximum when half the people are susceptible, and half the people are infectious for most iterations of the simulation as shown in Figure 6.7.

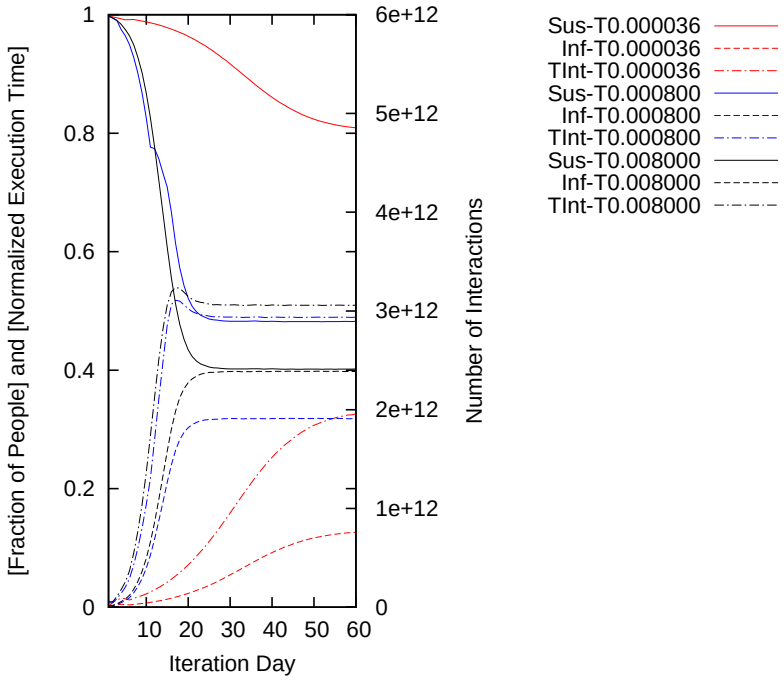


Figure 6.7 Shows the effects of transmissibility on people in the susceptible and infectious state, and the resulting potential number of interactions. The study was performed for transmissibility of 0.000036, 0.000080 and 0.00080. The number of potential interactions reaches a maximum when close to half of the population is infectious, and half is susceptible.

6.3.1.2 Quantification

Using the data from the experiment in Figure 6.6, we perform curve estimation and develop quantification equations. The equations will be able to estimate the compute times for all values of transmissibility and fixed values of the infectious period and incubation periods. For curve estimation, we use the statistical analysis tool [90], SPSS 22.0. This tool is useful in showing the qualify of fit of the equations. Looking at the figure, the compute time curves for all disease models show S-type curves. S-type curves can be estimated using exponential equations with two constants (b_0 and b_1). Its general form (for estimating effects of transmissibility on compute time) is given in Equation 6.3.

$$y = e^{(b_0 + b_1/\rho)} \tag{6.3}$$

Equation 6.3 can be used to estimate the compute time curves of Figure 6.6(a). We use the curve fitting feature of SPSS to extract its constants. Each disease model has its own set of constants as given in Table 6.4. The table shows that the estimation has an R-square (r^2) of 0.95, 0.98, and 0.97, for SIR, SIS and SI equations respectively, which means that the

equations can explain 95% of the simulation execution time of SIR, 98% of execution time of SIS, and 97% of execution time of SI, for all transmissibility and fixed values of infectious and incubation periods.

Table 6.4 Constants for the disease model equations that quantify the individual effects of disease transmissibility on compute time.

Model	ρ	Δt_I	Δt_E	b_0	b_1	r^2
SIR	0 – 1	4	1	8.399	−0.00039	0.98
SIS	0 – 1	4	1	4.716	−0.00003065	0.95
SI	0 – 1	∞[1]	1	4.497	−0.00001194	0.97

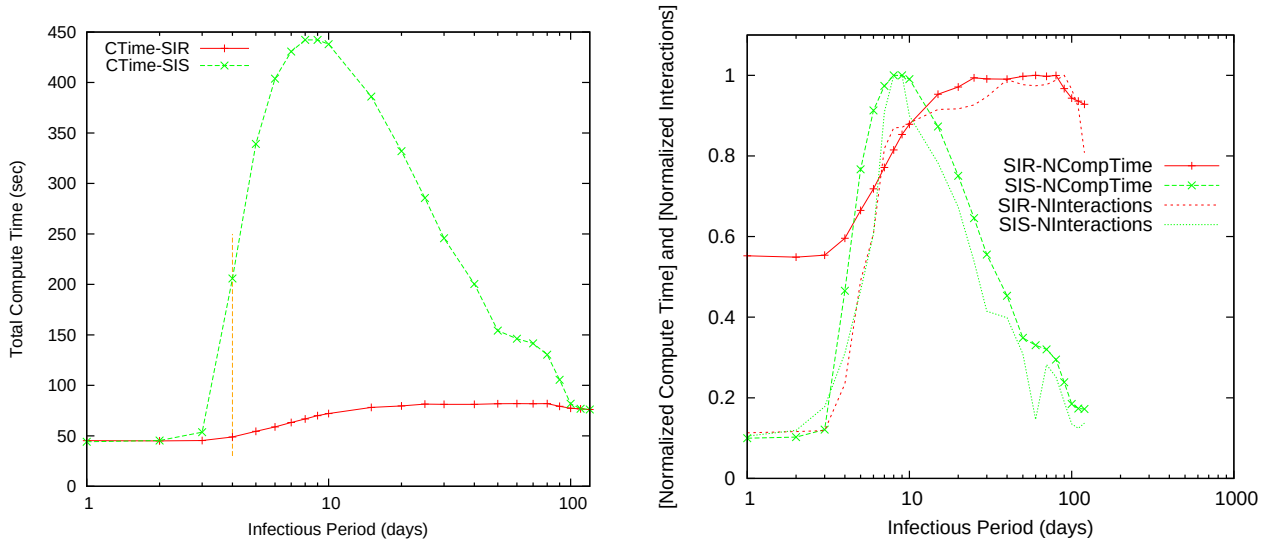
[1] fixed at simulation length

6.3.2 Effects of Infectious Period

6.3.2.1 Analysis

Infectious period (Δt_I) is the period during which an infected individual can transmit the disease to susceptible individuals. An increase in the infectious period keeps the individual in the infectious state for longer, which results in more infections being transmitted per person (high R_0), and may potentially increase the number of interactions. To study the effect of the infectious period on the execution time, we have performed an experiment with the setup introduced in Section 6.2. In this experiment, we keep the disease transmissibility and incubation period fixed at 0.000036 and one day, respectively, while varying the infectious period between 1 — 120 days (the length of the simulation). We choose to fix the transmissibility at 0.000036, because at this transmissibility, the compute time is middleway between the maximum and the minimum for all disease models; this is marked by the orange line in Figure 6.6(a). We choose an incubation period of one day, because it is not helpful to have a larger fraction of people in staying in the latent state. We use a time-step of 24 hours as shown in Section 3.1, and therefore, could not use an incubation period lesser than one day. As we perform further analysis of effects of the infectious period, the significance of these points will be further highlighted. We do not perform the experiment for SI, since in this case, the infectious period is fixed at infinity.

Figure 6.8 shows that when the infectious period is less than four days, the compute time and the number of potential interactions remain constant for both SIR and SIS. The reason for this effect is that in a short infectious period, the infectious people do not stay in the infectious state long enough to increase the fraction of infectious people (FI_r) to the point to cause an epidemic. In fact, the fraction of infectious people (FS_r) never goes beyond 1% (of total persons) at any time during the simulation. After the threshold, the compute time in the SIR model increases linearly with an increase in the infectious period. On the other



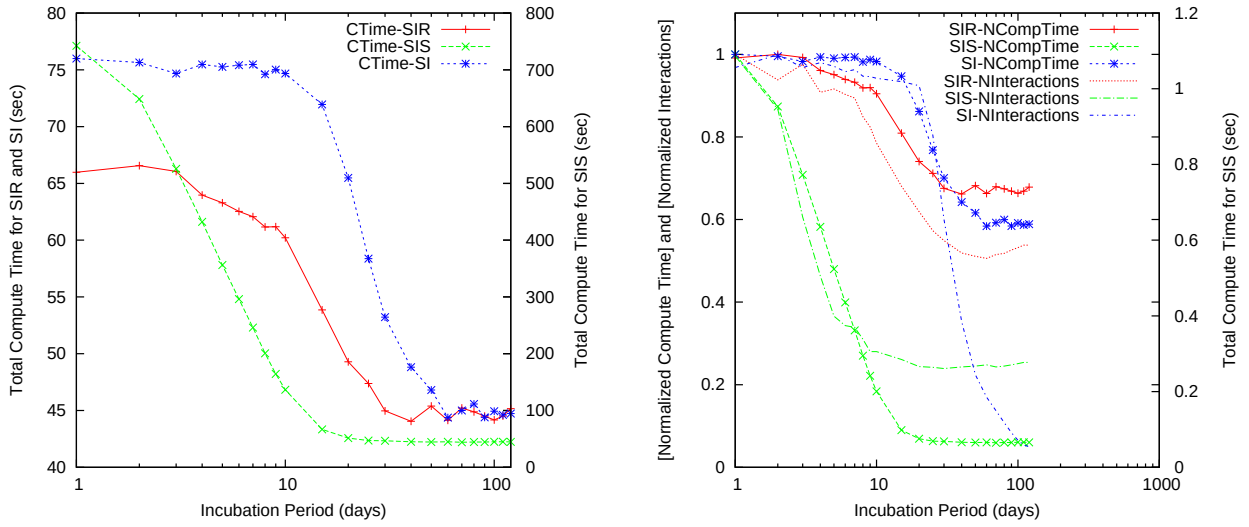
(a) **(b)**

Figure 6.8 Effects on an increase in infectious period on the compute time in SIR and SIS disease models. a) On a very small infectious period (1 – 3 days), the epidemic does not happen ($R_0 < 1$), thus resulting in smaller compute times (taken mostly by simulation for booking keeping) due to lesser or no interactions. b) For the infectious period more than 10 days, the compute time and potential number of interactions for SIS drops quickly, because the fraction of infectious individuals (FI_r) is much more compared to the fraction of susceptible individuals (FS_r).

hand, in the SIS model, the compute time increases exponentially with an increase in the number of interactions. This increase continues up to an infection period of 10 days, but starts decreasing after that. when the infectious period moves beyond 10, the fraction of infectious people (FI_r) becomes more compared to the fraction of susceptible individuals, resulting in fewer interactions, and hence smaller total compute times.

6.3.2.2 Quantification

After analyzing the effects of the infectious period, we quantify them using analytical equations — one equation per disease model. For such quantification, we use the SPSS curve estimation tool, and the data collected from the experiment in Figure 6.8. These equations will work for all values of the infectious period, and fixed values of transmissibility and incubation period. The SPSS curve estimation tool estimates the compute time curves using quadratic equations. Quadratic equations have three constants (b_0, b_1 and b_2) and its general form (for



(a) **(b)**

Figure 6.9 Larger incubation period results in an increased number of people in the exposed state and a lesser number of people in the susceptible and infectious state (smaller FS_r and FI_r), which further results in a reduced compute time and lesser potential interactions. This effect is higher in SIS, because the persons frequently get infected, and enter the incubation state.

estimating the effects of infectious period on compute time) is given in Equation 6.4.

$$y = b_0 + b_1 \Delta t_I + b_2 \Delta t_I^2 \tag{6.4}$$

Equation 6.4 estimates the compute time curves shown in Figure 6.8(a). The constants in the equations of SIR and SIS disease models are given in Table 6.5. The analysis shows that the estimation has an R-square (r^2) of 0.94 and 0.97 for SIR and SIS respectively, which means that the equation for SIR can explain 94% of the simulation execution time, and equation for SIS captures 97% of the simulation execution time. These equations are valid for all values of infectious period and fixed values of transmissibility and incubation period.

Table 6.5 Constants for the disease model equations that quantify the individual effects of the infectious period on the total compute time

Model	ρ	Δt_I	Δt_E	b_0	b_1	b_2	r^2
SIR	0.000036	1 – 120	1	45.699	−0.012	0.012	0.94
SIS	0.000036	1 – 120	1	38.312	0.987	−0.007	0.97

6.3.3 Effects of Incubation Period

6.3.3.1 Analysis

Incubation period (Δt_E) is the interval during which the infected individuals cannot transmit the disease to other susceptible individuals. In general, increase in the incubation period means a larger fraction of people in the incubation state, which means a smaller fraction of susceptible and infectious people, thus resulting in a lesser number of interactions. To study the effects of incubation duration on the execution time in more detail, we have performed some experiments using the experimental setup in Section 6.2. We keep the disease transmissibility and infectious period fixed at 0.000036 and four days, respectively, while varying the incubation period between 1 — 120 days (the length of the simulation). We choose the fixed transmissibility of 0.000036, because we noticed in Figure 6.6 that at this transmissibility the compute time is almost between the maximum and the minimum for all disease models. We choose an infectious duration of four days because as we see in Figure 6.8, given the transmissibility of 0.000036, the compute times are middleway between the maximum and the minimum for both SIR and SIS disease models.

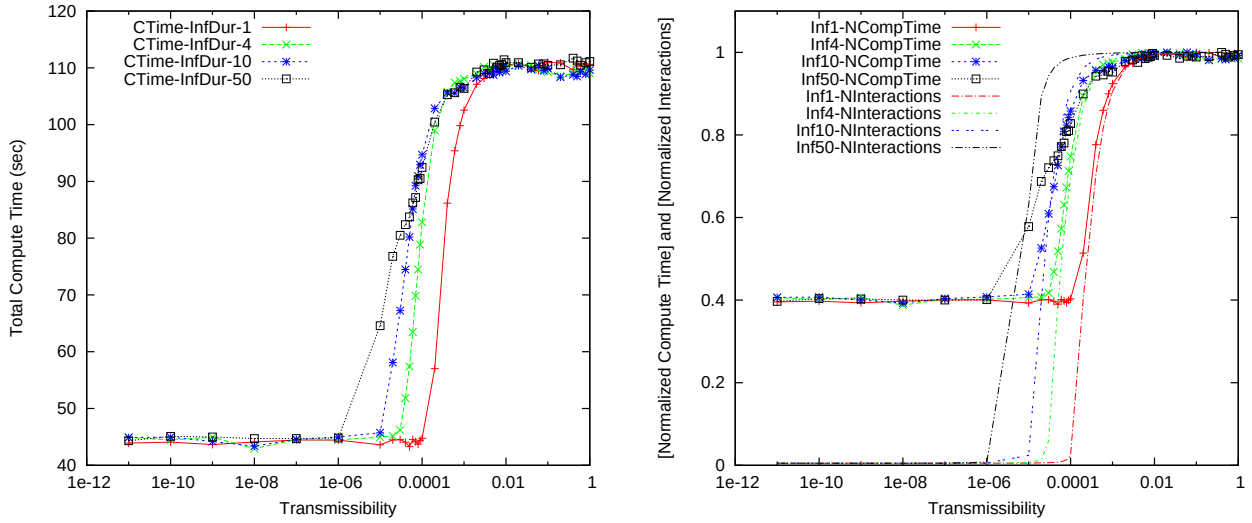
Figure 6.9 shows that an increase in the incubation period decreases the execution time and the number of interactions in all three disease models. The drop is exponential, but the slopes of the curves are different for different disease models. The slopes of SIS and SI are steeper compared to SIR. In SIS, people can get infected multiple times, and then enter into the incubation state for a longer period, which reduces the total number of interactions. In SI, even with a higher incubation period, the disease is still able to progress in later steps of the simulation, thus performing a good number of interactions.

In a very large incubation period, the fraction of people in the incubation state increases to the point where the epidemic does not actually occur. Again, this starting point is different for disease models as shown in Table 6.3.

6.3.3.2 Quantification

We extract those equations that quantify the individual effects of the incubation period using data from the experiment in Figure 6.9. The equations will be able to estimate the compute time for all values of the incubation period, and fixed values of transmissibility and infectious period. Looking at the figure, it can be said that the computation curves for all disease models show reverse S-type curves. The curves can be represented using an exponential equation as shown in Equation 6.5.

$$y = e^{(b_0 + b_1/\Delta t_E)} \tag{6.5}$$



(a) SIR-ComputeTime

(b) SIR-NormCTime+PInteractions

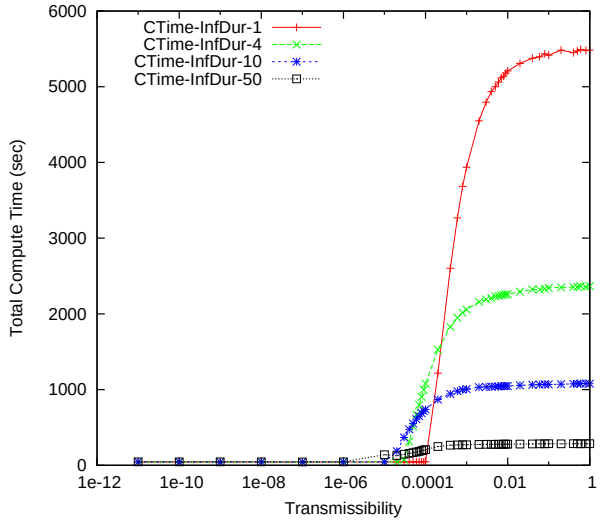
Figure 6.10 An increase in the infectious period does not change the shape of the execution time curves (S-type curves) of the SIR model, but lowers them by changing proportions of FS_r and FI_r. The curves refer to different values of infectious periods as labeled in the caption. The interaction curves, closely following the compute time curves, show that the compute time is largely dependent on the number of interactions.

The constants for these equations are given in Table 6.6. The analysis shows that the estimation has an R-square of 0.95, 0.96, and 0.98 for SIR, SIS, and SI respectively, which means that the equations can explain at least 95% of the simulation execution time of in all disease models, for all values of incubation period and fixed values of transmissibility and infectious period.

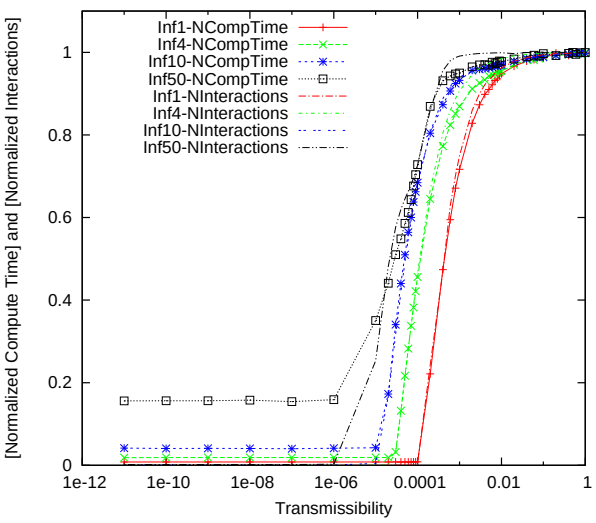
Table 6.6 Constants for the disease model equations that quantify the individual effects of the incubation period on compute time.

Model	ρ	Δt_I	Δt_E	b_0	b_1	r^2
SIR	0.000036	4	1 – 120	3.8	1.65	0.95
SIS	0.000036	4	1 – 120	3.8	0.110	0.96
SI	0.000036	∞[1]	1 – 120	4.004	0.604	0.98

[1] fixed at simulation length



(a) SIS-ComputeTime



(b) SIS-NormCTime+PInteractions

Figure 6.11 An increase in the infectious period does not change the shape of the execution time curves (S-type curves) for SIS, but lowers them by changing the proportions of FS_r and FI_r.

6.4 Joint Effects of Disease Model Parameters

In this section, we study the joint effects of disease model parameters on the compute time in the SIR, SIS, and SI disease models. We first study the pairwise effects of disease model parameters, i.e. variation in two disease model parameters while keeping the third parameter constant. Then, we study the effects of variation in all three disease model parameters and discuss their effects on the execution time. We also develop analytical equations (one per disease model) to quantify these effects.

6.4.1 Pairwise Effects of Transmissibility and Infectious Period

6.4.1.1 Analysis

To study the pairwise effects of transmissibility and infectious period, we have performed an experiment using the experimental setup discussed in Section 6.2. In the experiment, we study transmissibility in the range of 0.0 – 1.0, using four values of infectious period, 1, 4, 10, and 50 days for the SIR and SIS disease models. We choose the infectious duration of 1, 4, 10, 50 days from Figure 6.8 as these points show interesting ranges of compute time

curves. We do not perform the experiment for SI, since the infectious period is fixed at infinity. Figure 6.10 shows that the compute time curves for SIR lowers as the infectious period increases from 1 to 50 days. This means that an increase in the infectious period reduces the compute time. We see a similar effect for SIS as shown in Figure 6.11. This was expected, because the increase in the infectious duration forces more people to stay in the infectious state, thus decreasing the fraction of people in the suseptible state (FS_r), which results in a lesser number of interactions. The decrease is more visible in SIS than SIR. And the reason for this effect is that the SIS compute times vary in a much larger range.

6.4.1.2 Quantification

After analyzing the pairwise effects of transmissibility and infectious duration, we use data from the experiments to perform curve estimation. The equations will be able to estimate the compute times for all values of transmissibility and infectious period, and fixed values of incubation period. Again, we use the statistical analysis tool [90], SPSS version 22. Looking at Figure 6.10 and 6.11, we see that the compute time curves for all disease models are S-type curves. These S-type curves can be estimated by exponential equations with two constants (b_0 and b_1). The cost estimation metric is given in Equation 6.6.

$$TCOMP_{\rho,\Delta t_I} = \frac{e^{(b_0+b_1/\rho)}}{(b_3 \Delta t_I)} \tag{6.6}$$

The constants for SIR and SIS equations are shown in Table 6.7.

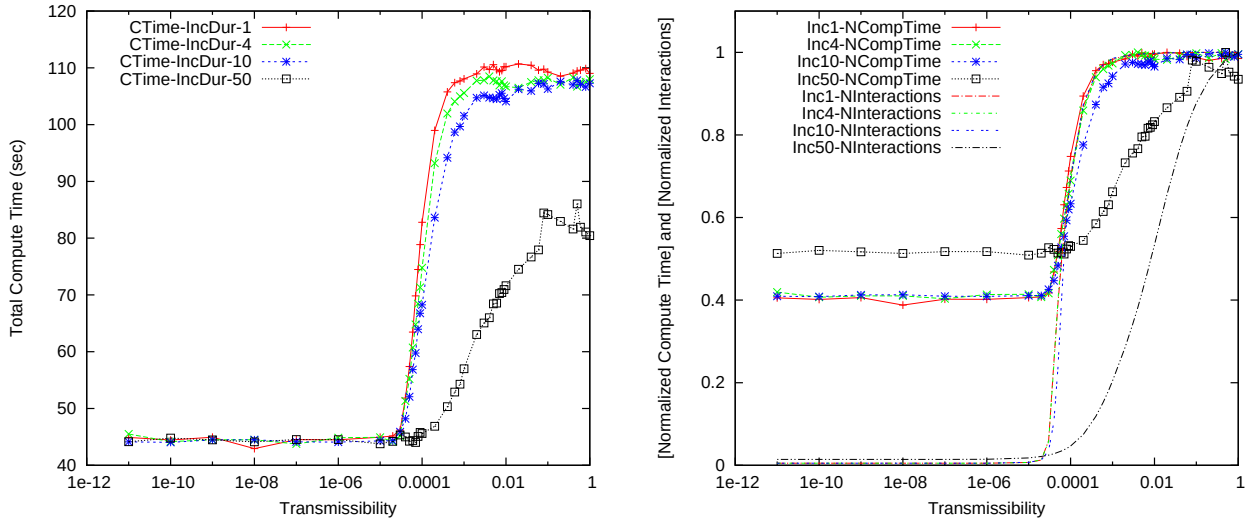
Table 6.7 Constants for the disease model equations that quantify the pairwise effects of disease transmissibility and infectious period on compute time.

Model	ρ	Δt_I	Δt_E	b_0	b_1	b_3
SIR	0 – 1	1 – 120	1	7.737	−0.000075	0.063
SIS	0 – 1	1 – 120	1	4.706	−0.000016	0.112

6.4.2 Pairwise Effects of Transmissibility and Incubation Period

6.4.2.1 Analysis

In this section, we study the pairwise effect of transmissibility and incubation period on the execution time curves. We vary transmissibility in the range of 0.0 — 1.0, while checking for four values of the incubation period — 1, 4, 10 and 50 days for all disease models. The incubation period of 1, 4, 10, and 50 were chosen based on the significance of these points on



(a) SIR-ComputeTime

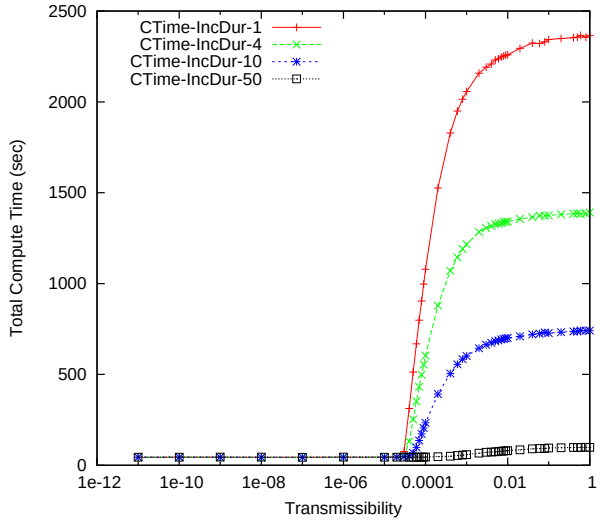
(b) SIR-NormCTime+PInteractions

Figure 6.12 An increase in the incubation period increases the proportion of individuals in the incubation state, and decreases the fraction of infectious and susceptible individuals (FS_r and FI_r), thus resulting in lowered execution time curves. The constant compute time at lower ranges of transmissibility is the time taken by the simulation when no interactions occur.

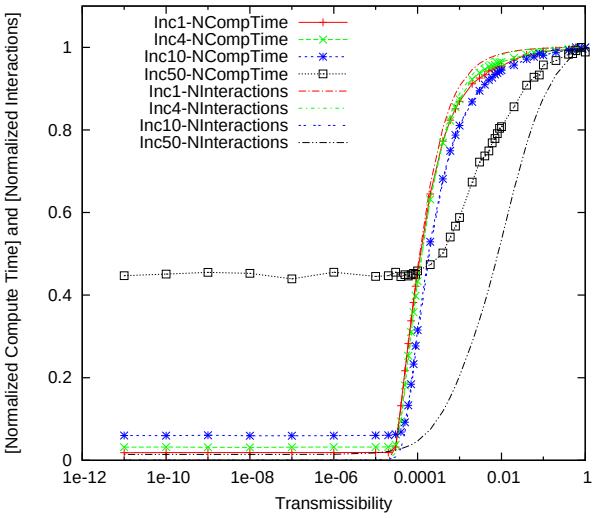
the compute times in Figure 6.9. As expected, the compute time curves for all three disease models are lowered as the incubation period is increased from 1 to 50 days, as shown in Figure 6.12, 6.13, and 6.14, for SIR, SIS and SI respectively. The decrease is more in SIS, as its compute times vary in a much larger range.

6.4.2.2 Quantification

After analyzing the pairwise effects of transmissibility and incubation period, we use the data collected from the experiment to perform curve estimation in order to develop quantification equations. The equations will be able to estimate the compute time for all values of transmissibility and incubation period, and the fixed value of infectious period. Looking at the figure, the compute time curves for all disease models show reverse S-type curves. Such S-type curves can be estimated using exponential equations with two constants (b_0 and b_1). The Equation 6.7 shows the quantification for all disease models. Using SPSS, we added one more constant to the denominator of the equation in order to make it estimate the execution



(a) SIS-ComputeTime



(b) SIS-NormCTime+PInteractions

Figure 6.13 An increase in the incubation period increases the proportion of individuals in the incubation state, and decreases the fraction of infectious and susceptible individuals, resulting in lowered execution time curves.

times for all the values of the incubation period.

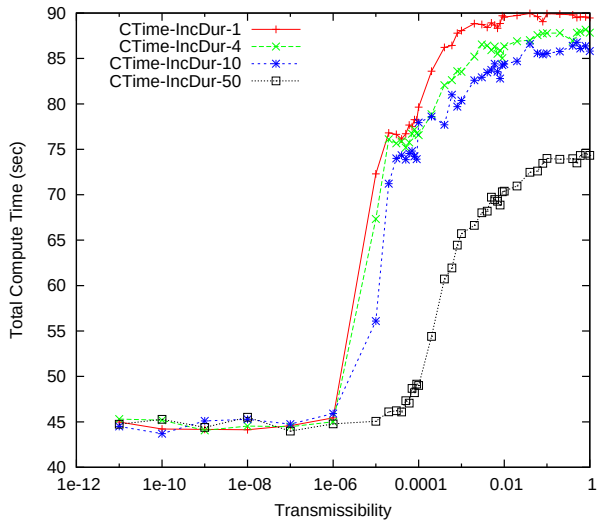
$$TCOMP_{\rho,\Delta t_E} = \frac{e^{(b_0+b_1/\rho)}}{(b_4 \Delta t_E)} \tag{6.7}$$

The constants for the equations of SIR, SIS and SI are shown in Table 6.8.

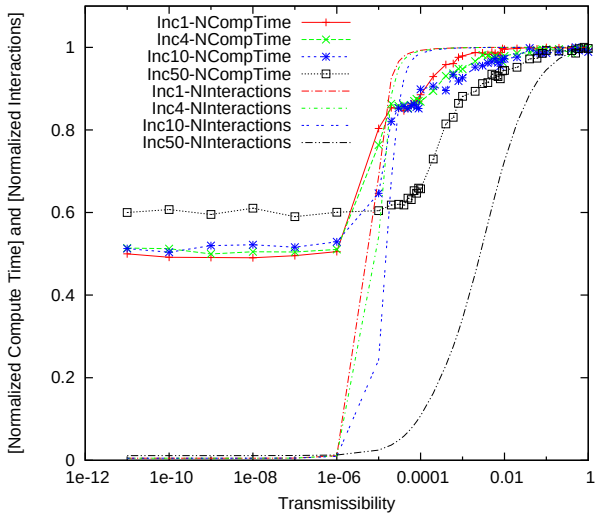
Table 6.8 Constants for the disease model equations that quantify the pairwise effects of disease transmissibility and incubation period on compute time.

Model	ρ	Δt_I	Δt_E	b_0	b_1	b_4
SIR	0 – 1	4	1 – 120	7.737	−0.00002312	0.091
SIS	0 – 1	4	1 – 120	4.686	−0.00003411	0.450
SI	0 – 1	∞[1]	1 – 120	4.686	−0.00003411	0.090

[1] fixed at simulation length



(a) SI-ComputeTime



(b) SI-NormCTime+PInteractions

Figure 6.14 An increase in the incubation period increases the proportion of individuals in the incubation state, and decreases the fraction of infectious and susceptible individuals, resulting in lowered execution time curves.

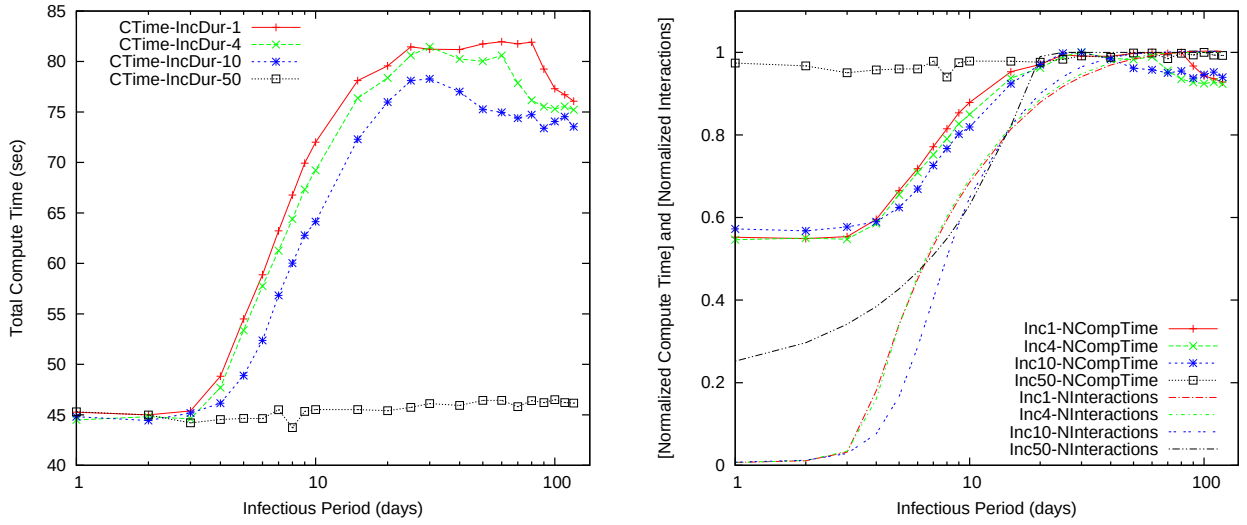
6.4.3 Pairwise Effects of Incubation and Infectious Period

6.4.3.1 Analysis

We also study the pairwise effects of infectious and incubation period on the execution time. We vary infectious period in the range of 1 — 120, while checking for four values of incubation period — 1, 4, 10 and 50 days for all disease models. As expected, increase in the incubation period lowers the compute time curves for both SIR and SIS disease models as shown in Figure 6.15 and 6.16. The decrease is more visible in SIS, as its compute times vary in a much larger range. This analysis is not applicable to SI, as this model's infectious period is fixed at infinity.

6.4.3.2 Quantification

After analyzing the pairwise effects of the infectious period and incubation period, we use the data from the experiment to develop the quantification equations. These equations will be able to estimate the compute time for all values of the infectious and incubation period, and fixed values of transmissibility. Looking at the figure, the compute time curves for all



(a) SIR-ComputeTime

(b) SIR-NormCTime+PInteractions

Figure 6.15 An increase in the incubation period means an increase in the presence of people in the latent period, which results in a lesser number of interactions and the lowering of compute time curves.

the disease models show quadratic curves. These quadratic equations have three constants (b_0, b_1 and b_2). The compute times of SIR, SIS, and SI are quantified using Equation 6.8.

$$TCOMP_{\Delta t_I, \Delta t_E} = \frac{b_0 + b_1 \Delta t_I + b_2 \Delta t_I^2}{(b_5 \Delta t_E)} \tag{6.8}$$

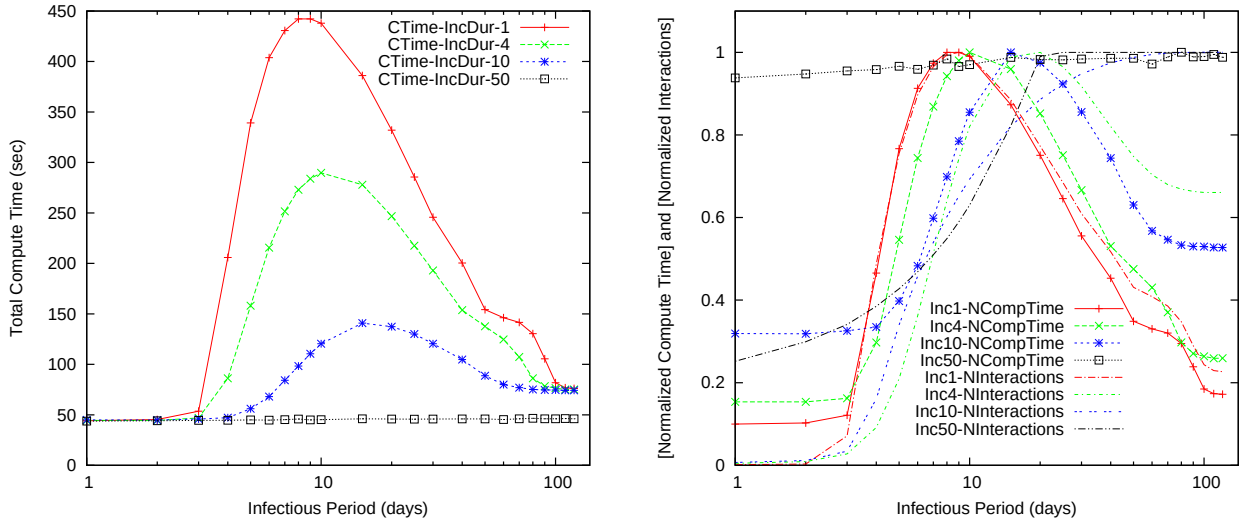
The constants for the equations are shown in Table 6.9.

Table 6.9 Constants for the disease model equations that quantify the pairwise effects of infectious period and incubation period on compute time.

Model	ρ	Δt_I	Δt_E	b_0	b_1	b_2	b_5
SIR	0.000036	0 – 1	1 – 120	48.33	0.307	−0.0032	0.11
SIS	0.000036	0 – 1	1 – 120	48.33	−0.032	0.0500	0.63

6.4.4 Joint Effects of all Disease Model Parameters

In this section, we quantify the effects of variation in all three disease model parameters on the compute time in the SIR, SIS, and SI disease models. Figure 6.17 shows the effect



(a) SIS-ComputeTime

(b) SIS-NormCTime+PInteractions

Figure 6.16 An increase in the incubation period means a higher number of people in the latent period, which results in a lesser number of interactions and the lowering of compute time curves.

of transmissibility and infectious period for four values of incubation period. Increasing transmissibility increases the compute time by changing the rate of transfer of people from susceptible to infectious state. But increasing the infectious state accumulate more people in the infectious state which reduces the number of interaction, and hence compute time. Similarly, increases the incubation state increases the proportion of people in the incuabtions state (which do not perform interactions), and reduces the compute time.

We use the experimental data from Section 6.3 and Section 6.4, which was also used for the analysis of individual and pairwise effects in order to extract an equation that can capture the dynamics of these disease models.

Again, we use the SPSS curve fitting tool to determine the equation. The equation produced by curve fitting has an R-square r^2 of about 0.80 for all three models, which means that the independent variables (transmissibility, infectious period, incubation period) can explain 85% of the compute time in all disease models. To improve the r^2 further, we apply the multi-variable regression analysis. In this regression modeling, we add and remove linear, quadratic, and exponential components of independent variables, and see their effect on the r^2. If a term is statistically significant and improves the r^2, it is added to the equation; otherwise, it is ignored.

Table 6.10 Constants (k's) for the dynamic quantification equation ($Tcost$) of SIR, SIS and SI disease models.

Model	k_0	k_1	k_2	k_3	k_4	k_5	k_6	r^2
SIR	49.21	0.308	0.565	0	0.001	0.125	0.095	0.85
SIS	48.96	0.308	0.308	0	0.510	−0.001	0.450	0.96
SI	49.72	0.403	0	6.925	0	0.403	0	0.91

Table 6.11 Constants (b's) for the dynamic quantification equation ($Tcost$) of SIR, SIS and SI disease models.

Model	b_1	b_2	b_3	b_4
SIR	4.7	0.000039	8.4899	0.00039
SIS	8.4899	0.00039	9.899	0.00039
SI	4.7	0.000011	0	0

The final equation for SIR, SIS, and SI disease model is shown through Equation 6.9.

$$DQ = k_0 + k_1 e^{b_0 + b_1/\rho} + k_2 \frac{e^{b_3 + b_4/\rho}}{k_6 \Delta t_E + {\Delta t_I}^{0.5}} + k_4 {\Delta t_I}^2 + k_5 \Delta t_E \tag{6.9}$$

The constants for equation are given in Tables 6.10 and 6.11. Note that some of the constants are zero for some of the disease models, which means that not all the components of the equation are used for every disease model. The higher values of r^2 in Table 6.10 show that the equation captures the effects of the disease model parameters on the compute time in a good proportion.

6.5 Disease Dynamics and the Cost Metric

In this section, we combine the static quantification equation (developed in Chapter 4) and the equations that capture the dynamics of disease models (developed in this chapter) in order to develop another equation that estimates the execution cost of EpiSimdemics. The static cost equation was useful in estimating the execution times of EpiSimdemics for different data sets on different numbers of processors with a maximum of 6.5% error. However, this equation does not have the capability to capture the variability in compute times that are caused by the dynamics of the disease models, affected by the changes made to transmissibility, the infectious period and the incubation period. Therefore, we want to combine the static equation and the equation of disease dynamics in order to compute the total execution cost of the simulation. The resulting metric should be able to capture the variability in data sizes, processor counts, and disease models.

From Table 6.10, we can understand that the constant intercepts (k_0) are almost the same (49.29 ± 0.43) for all three equations. When we compute the static cost (*Mcost*) for these configurations, they are also very close to the constant intercept numbers (48.75 ± 0.81). Therefore, we replace the constant intercept in *DQ* (Equation 6.9) with *Mcost* in order to formulate an equation for the total execution cost (*Tcost*). The resultant *Tcost* equation is given in Equations 6.10, which can compute the execution times of EpiSimdemics for any configuration.

$$Tcost = Mcost + k_1 e^{b_0 + b_1/\rho} + k_2 \frac{e^{b_3 + b_4/\rho}}{k_6 \Delta t_E + {\Delta t_I}^{0.5}} + k_4 {\Delta t_I}^2 + k_5 \Delta t_E \tag{6.10}$$

The constants for the equation are given in Tables 6.10 and 6.11.

In *Tcost*, the load balancing (load assignment) variation is covered by *Mcost* and the quantification of disease dynamics is covered by the *DQ* equations.

6.5.1 Effectiveness of the Cost Metric

To test if the cost metric (*Tcost*) estimates the execution times in a good proportion (that covers data sizes, processor counts, and disease dynamics), we perform certain experiments. In these experiments, we choose three data sets: AL (Alabama), NC (North Carolina), and IL (Illinois). AL is our smallest dataset and was utilized in the extraction of static and disease dynamic equations. NC is larger than AL, and was used in extraction of static quantification equation. Further, IL is larger than AL and NC, and it will be interesting to see its estimation error. We collect the execution times on 60, 96, and 120 processors. We compare the actual execution times against the predicted execution times (*Tcost*) and analyze the mean and variance of absolute estimation error.

Firstly, We compare the actual execution times and the estimated execution times for all data. Figures 6.18 and 6.19 show that the cost metric have estimated the execution times across all the levels (of execution times) in a good proportion. The average error is 16.13%, which is an indication of the *Tcost* estimation being good, given that the error is computed across a processor count with a variety of data on different disease models. In Section 6.5.1.4, we offer the reasons for considering 16.13% to be a reasonable estimation error.

Secondly, we show the ability of the cost estimation metric to predict execution times across different configurations (processor counts, data sizes, and disease models).

6.5.1.1 Variation in Data Sizes

We begin our evaluation with the AL dataset. Since this dataset was used in the development of *Mcost* and *DQ* equations, we expect a good estimation for all of its configurations.

Figure 6.20 (a) shows that the cost equation has estimated the execution times in a good proportion in all levels of experimental data. Table 6.12 shows that the mean and standard deviation of absolute error is only 15.01% and 15.3% respectively.

Table 6.12 Estimation error of *Tcost* when predicting execution times of different data sets.

Data	Min.	Max.	Abs. Mean	Variance	Std. Dev.
AL	0.022	90.52	15.01	177.36	13.3
NC	0.0243	38.47	17.9	205.72	14.5
IL	0.001	71.16	18.47	249.38	15.8

The *DQ* part of the *Tcost* equation is specific to AL and is not directly applicable to other data (i.e., NC and AL). To make it work on different data, we have added another parameter to the equation of disease dynamics. We have determined that if we adjust the *DQ* equation (Equation 6.9) by *Mcost*-estimated compute times, the estimation can be greatly improved. The computation in Equation 6.11 shows this adjustment.

$$DQAdj = DQ\frac{curMcostExecutionTime}{ALMcostExecutionTime} \tag{6.11}$$

where the *curMcostExecutionTime* is the *Mcost*-estimated execution time of that configuration, while *ALMcostExecutionTime* is the *Mcost*-estimated execution time of AL data.

After the adjustment, the cost metric estimates the execution times of the NC and IL data in a good proportion. The average absolute error for the NC and IL data is 17.9% and 18.47% respectively. The standard deviation of absolute error is 14.5% and 15.8% for NC and IL respectively.

6.5.1.2 Variation in Processor Counts

We also check the ability of the cost metric for different processor counts. Figure 6.21 shows that the cost metric has estimated the execution times of AL, NC, and IL data on 60, 96, and 120 processors across all the levels (of execution times) in good proportion. The absolute average error as shown in Table 6.13 is only 15.94%, 17.35%, and 16.91% for 60, 96, and 120 processor elements respectively. This shows the ability of the cost metric in estimating the execution times on different processor counts.

6.5.1.3 Dynamics of Disease Models

Finally, we perform an evaluation of the cost metric estimation on SIR, SIS, and SI disease models. Figure 6.22 shows that the cost estimation metric estimates the execution times for

Table 6.13 Estimation error of *Tcost* when predicting execution times at different number of processors.

nPEs	Min.	Max.	Abs. Mean	Variance	Std. Dev.
60	0.012	54.60	15.94	195.06	13.96
96	0.0643	90.52	17.03	285.72	16.90
120	0.01	42.91	16.91	249.38	15.8

Table 6.14 Estimation error of *Tcost* when estimating execution times of different disease models.

Model	Min.	Max.	Abs. Mean Error	Variance	Std. Dev.
SIR	0.00061	71.66	18.37%	332.82	18.24
SIS	0.21	90.52	14.16%	230.80	15.19
SI	0.41	32.85	12.90%	298.20	17.26

SIR and SIS across all the levels of compute times. However, in the case of SI, due to its small sample size, we do not see a good estimation across all the levels. But, the average absolute error is only 12.90%, which is the best of all three models. For the other two models, SIR and SI, the average absolute error is 18.37% and 14.17% respectively, which is shown in Table 6.14.

6.5.1.4 Reasons for Cost Estimation Error

In this section, we outline the possible reasons for high cost estimation error. Among many reasons, the two most important application-specific reasons are the network structure and the stochastic nature of application.

EpiSimdemics is run on the social contact network, which is a bipartite graph of persons and locations as shown in Figure 3.1 on page 13. The structure of this contact network (number of people, average number of contacts, household size, and age distribution, etc.) influences the spread of infectious disease in population as shown by Jiangzhuo et al. [102]. This point was also demonstrated by Taylor, Marathe and Beckman in their study dealing with the effects of vaccination strategies across the US cities [103]. They compared two metropolitan areas, namely Seattle and Miami, and proved that the effects of the same vaccination strategy were different in each of these cities. They argued that the regions differ significantly in terms of people's age and household size distributions — this played a huge role in the performance of the vaccination strategy.

Additionally, the structural dynamics for AL, NC, IL, and NY networks are shown in Table 6.15. The table shows that the networks are different from each other in terms of average edge degree, maximum edge degree and radius.

The cost estimation metric (*Tcost*) developed in this work does not consider the dynamics of the network structure, and therefore, let us anticipate that most of our cost estimation error is because of this factor. We further agree that addition of network dynamics feature to the cost metric will be a good enhancement and will greatly reduce the estimation error. Hence, we have listed the cost metric extension with network dynamics as one of our future works.

The stochastic nature of the application requires the researchers to use different random streams when running multiple replicates of the same configuration. Using such different random streams may introduce variation in the execution times. The random streams may affect the outbreak of disease and, as a result, the number of interactions. To highlight the variation from random streams in more detail, we have performed an experiment. In this experiment, we first chose 10 configurations. The configurations vary in transmissibility, infectious period, and incubation period for the three disease models. Then, we run 15 replicates of each configuration with different random stream seeds for each replica. Figure 6.23 shows that when using different random stream seeds, the execution times vary significantly. The normalized execution times vary across replicas in all configurations. In this figure, the location of the median point across configurations show that the execution times vary randomly as the seed is changed.

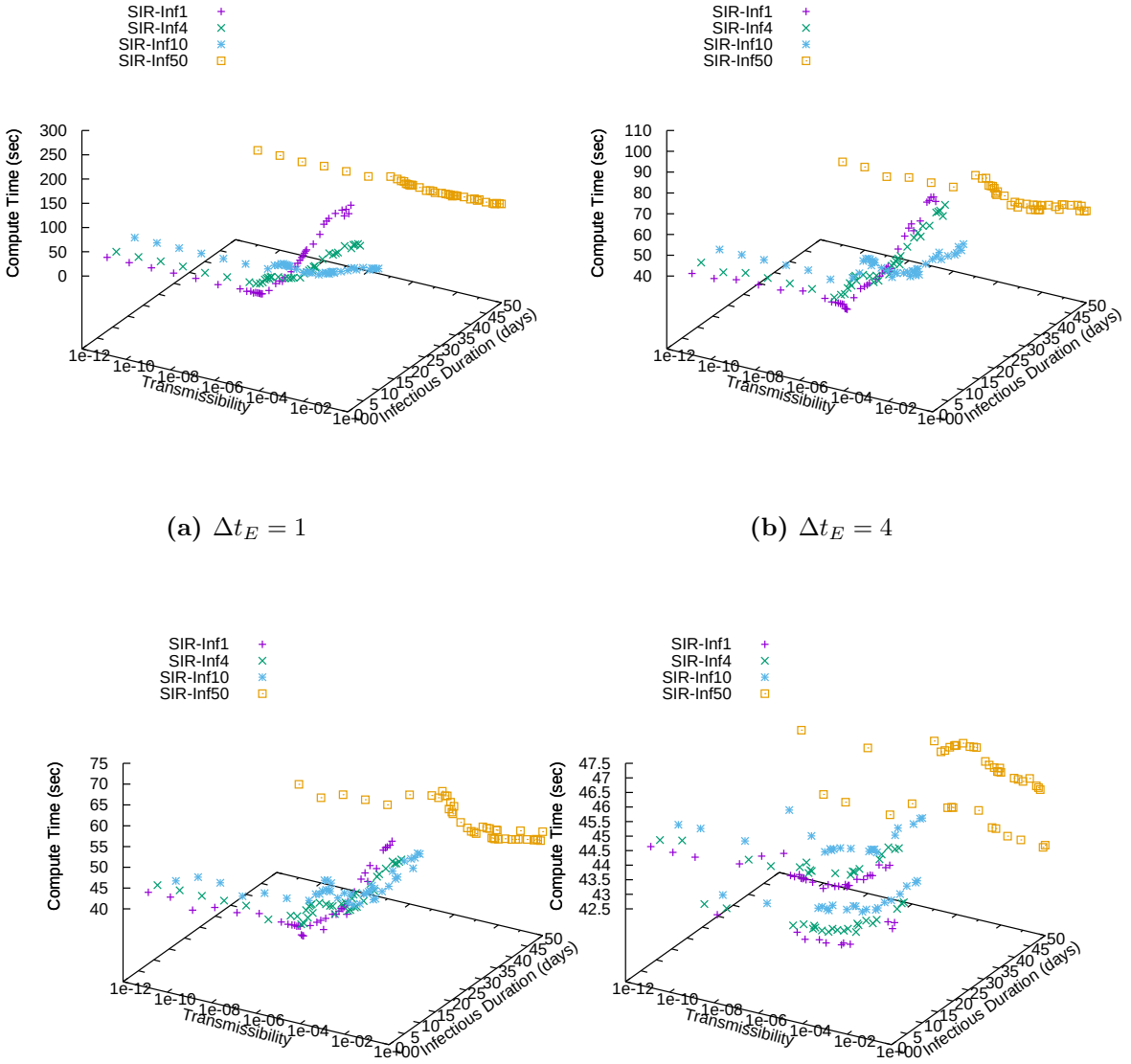
Our cost estimation does not cover this variation from the random streams, which will likely result in an estimation error. Furthermore, the variation is random in all the disease models, and could not be computed using simple methods. An additional study is required to capture the dynamics of random streams in different disease models.

Table 6.15 Structural properties of our social contact networks.

Network	Nodes	Edges	Avg. Deg.	Max Deg.	Min Deg.	Radius
AL	5572153	24127393	20.12	44230	1	5
NC	10830734	47130621	20.58	46505	1	5
IL	15018992	66143870	22.38	107225	1	4
NY	22630391	98350858	20.83	100387	1	4

6.6 Chapter Summary

In this chapter, we have analyzed the individual and joint effects of the disease model parameters (transmissibility, infectious period, and incubation period) on the execution time in the SIR, SIS, and SI disease models. Based on the analysis, we have developed analytical equations (one for each disease model) that quantifies these effects. The static and dynamic equation in combination estimates the execution times of different data on a number of processors for all disease models.



(a) $\Delta t_E = 1$ **(b)** $\Delta t_E = 4$

(c) $\Delta t_E = 10$ **(d)** $\Delta t_E = 50$

Figure 6.17 Shows the effect of disease model parameters on execution time in SIS disease model. Increase in infectious period results in accumulation of more people in the infectious state, which decreases the number of interactions (and computations). Increase in incubation period increases the proportion of people in incubation state, which do not perform interactions, hence reducing the total compute time.

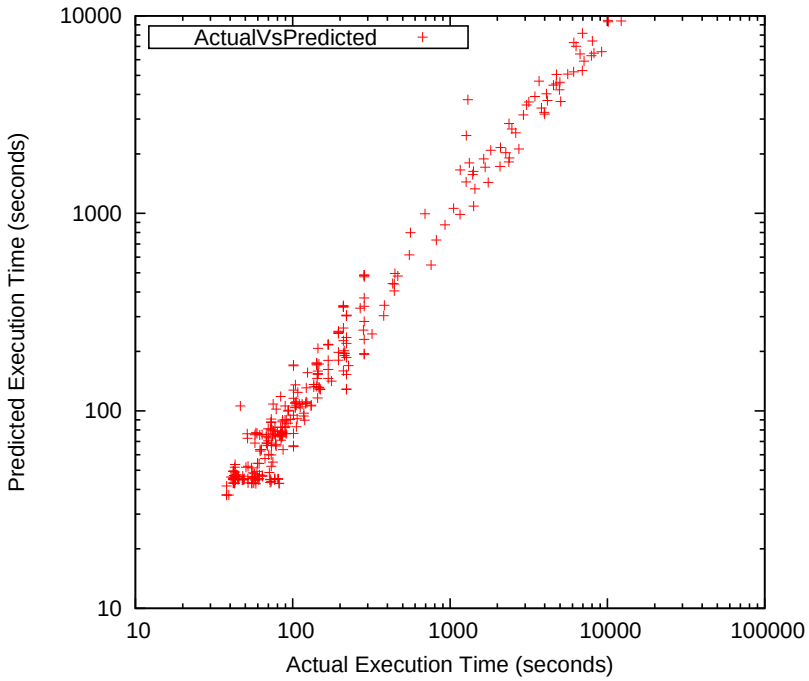


Figure 6.18 The cost estimation metric can estimate the execution times across a range of compute times. Average absolute percent errors and standard deviation across all the configurations is 16.13% and 16.65% respectively.

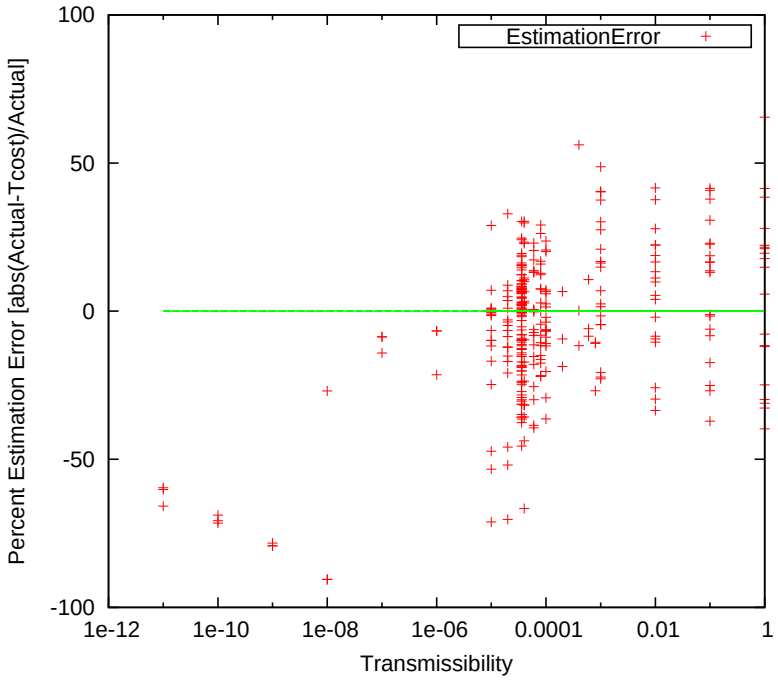
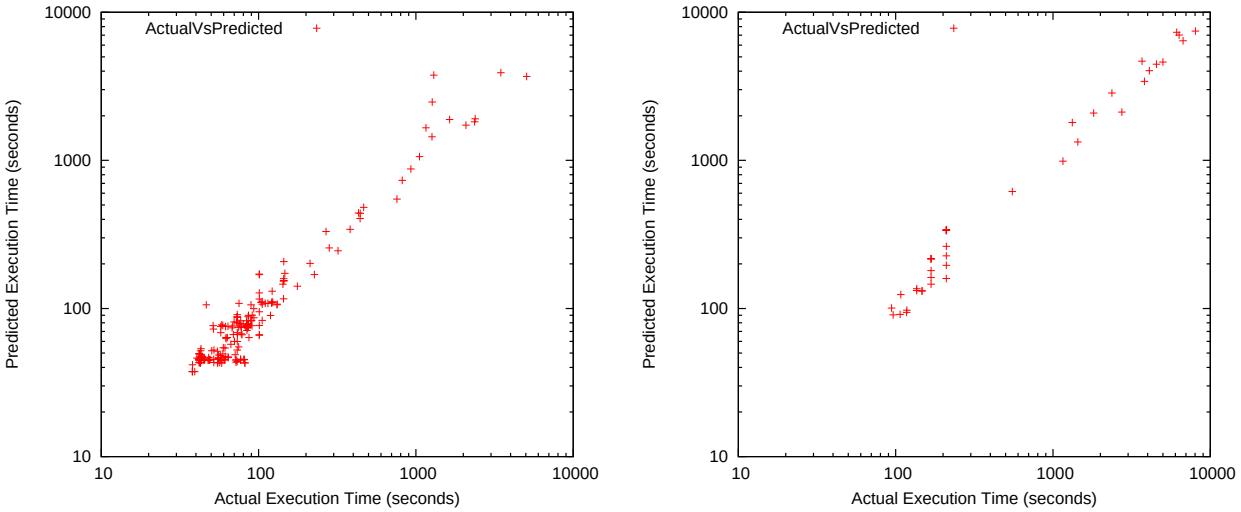
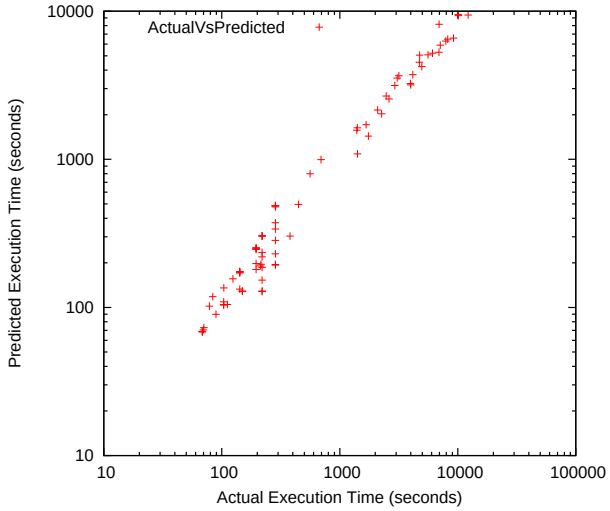


Figure 6.19 High error on the first few configurations shows that the equations have weak applicability at very small transmissibility (transmissibility $< 10^{-6}$).



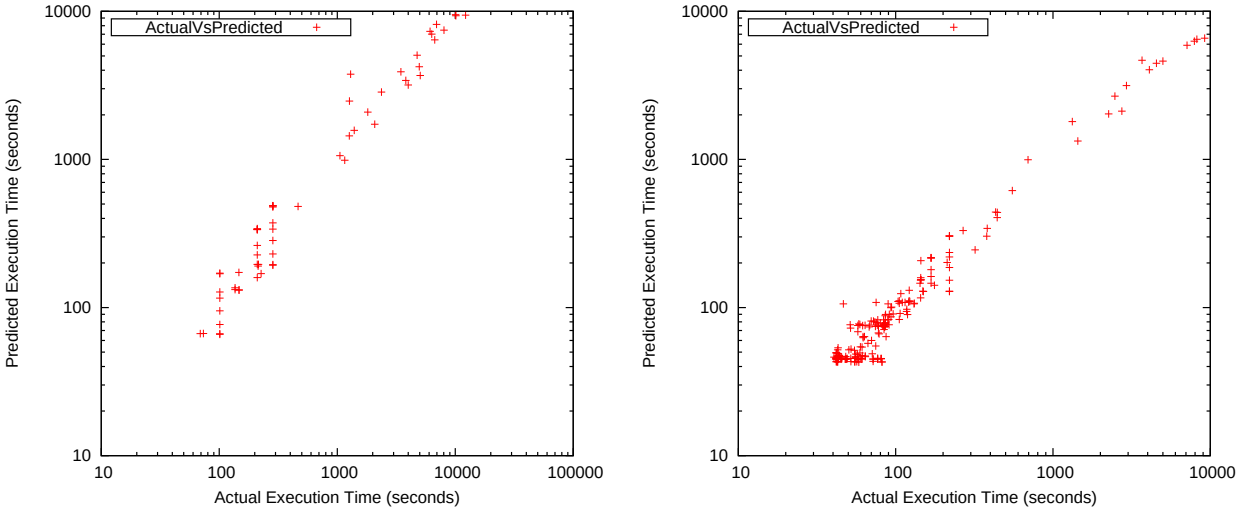
(a) AL

(b) NC



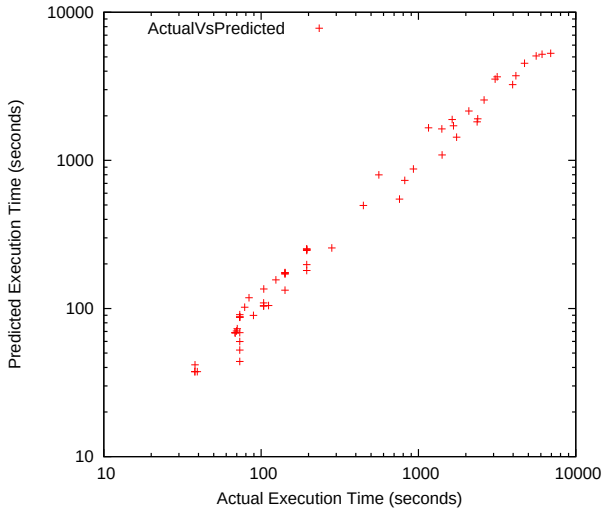
(c) IL

Figure 6.20 The cost estimation across all the levels shows that the metric is well applicable across different datasets. The error is slightly more for the NC and IL data, compared to AL, because the cost model is developed using the AL data. The average absolute percent error (a) for AL is only 15.01% (b) for NC 17.9%, and (c) for IL is 18.47%.



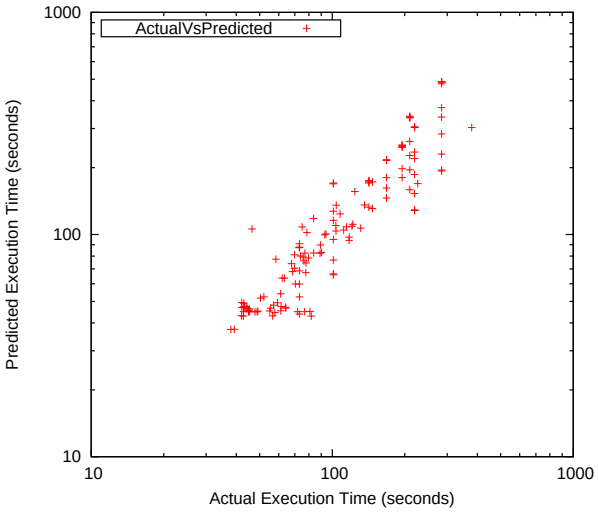
(a) 60 PEs

(b) 96 PEs

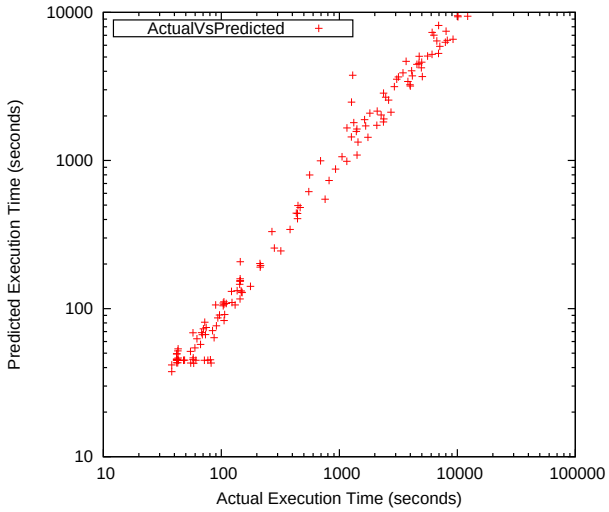


(c) 120 PEs

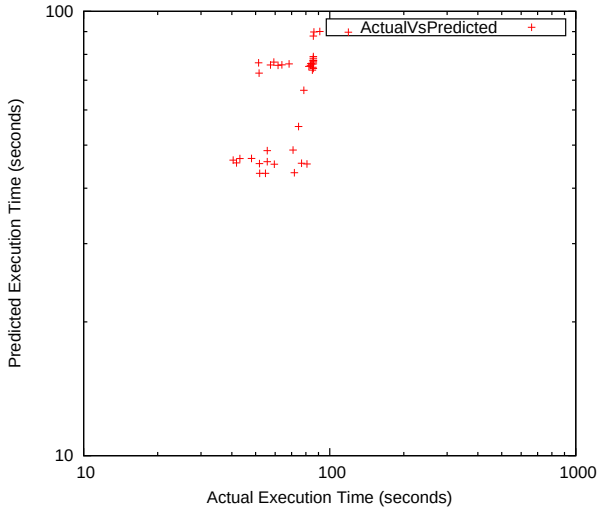
Figure 6.21 The cost metric estimates the execution times for different processor counts. Average absolute percent error when using (a) 60 PEs is 15.94% (b) when using 96 PEs is 17.35% (c) and at 120 PEs is 16.91%.



(a) SIR



(b) SIS



(c) SI

Figure 6.22 Cost estimation across the different levels of compute times shows that the model is general and well applicable to all three disease models.

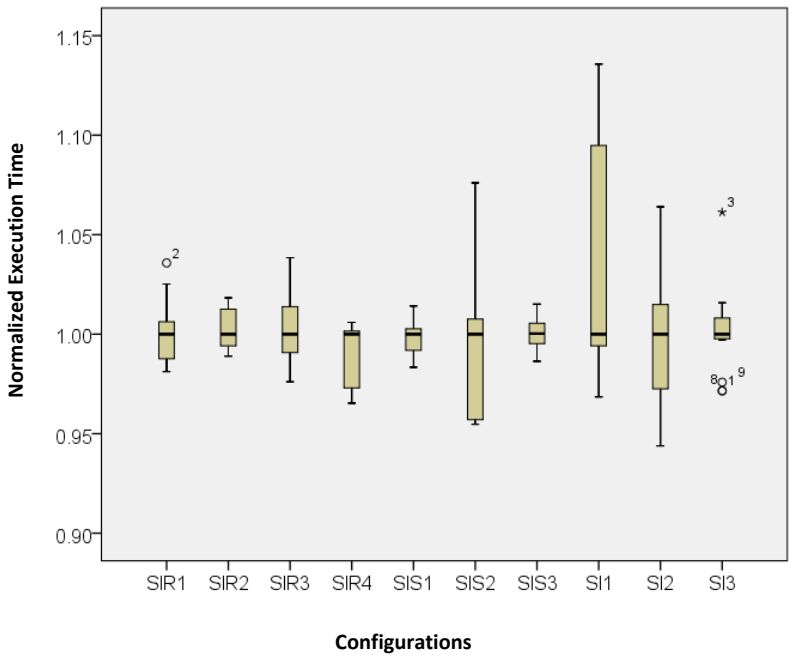


Figure 6.23 Box plot for random streams experiment. Across configurations, the execution time varies significantly for different random stream seeds (15 replicates of each configuration).

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This dissertation is aimed to perform the computational cost analysis of agent-based epidemic simulations (ABES) to improve their scalability (efficiency) on high-performance clusters. Owing to that, this work has achieved its purpose in four steps: performing static load analysis and quantification of computational and communication workloads: developing a cost estimation metric: studying and developing the static load distribution schemes: and analyzing and quantifying the effects of disease model parameters on the execution time. With respect to the static load analysis, we studied the input agent-location bipartite graph and the semantic of simulation algorithms. Our analysis showed that the agent computational load can be quantified using its outgoing edges, and the location computational load can be statically quantified using its incoming edges. The communication was quantified by using the inter-process communication between the agent and location objects.

To perform the performance modeling, resource allocation, and load distribution of any application, it is important to develop a cost estimation metric that can estimate the application's execution cost. For this purpose, we have developed a cost estimation metric that estimates the execution cost of agent-based epidemic simulations on high-performance clusters. This metric is generally applicable to all constrained producer-consumer algorithms. The cost metric follows the min-max principles and computes the execution cost using computational imbalances and communication latencies. Although, constants of the cost metric are application and machine-specific, we developed a detailed scientific methodology to extract them. Our evaluations show that the developed metric estimates the execution times of different data with a maximum of 6.5% error. Furthermore, the metric is useful in performance prediction, resource allocation, and the evaluation of load balancing schemes.

Moreover, intelligent data distribution plays a vital role in the scalability of parallel algorithms on high-performance clusters. To address this issue, we have adopted Metis, a well-known graph partitioning library, for the purpose of partitioning agent-location bipartite graphs. Our analysis shows that the Metis partitioned graphs achieve a two-fold performance gain compared to the Round-robin distribution scheme. However, its partitioning overhead

(runtime and memory) increases exponentially, when using larger data and creating a larger number of partitions. To mitigate this, we have created a three-layered custom streaming scheme that assigns agents and locations to processors. More specifically, the algorithm assigns locations to processors on the in-edges and then co-locates agents with their home locations. This is important from the communication perspective, because on average, 50% of the time the agents visit their home locations. The custom scheme performs better in comparison to Metis and has an extremely lower overhead.

The agent-based epidemic models are iterative and follow graph dynamical systems, where the computational and communication workload may change through iterations due to a change in the state of agents. This makes the cost estimation very challenging. To address this issue, we have analyzed the individual and joint effects of disease model parameters (transmissibility, infection period, and incubation period) on the compute times of the SIR, SIS, and SI disease models. We have also developed an analytical equation (with different constants for different disease models) to quantify such dynamics. To create dynamics-aware cost equations, we have merged the static min-max cost metric with dynamic-aware analytical equations. The resulting equations are more general and can work for different datasets, processor counts, and disease models. Our evaluations show that the merged cost metric estimates execution times with a maximum of 18% error. While further investigations are needed, most of the error appear to come from the dynamics of networks.

7.2 Limitations and Weaknesses of Our Approach

Although, our load analysis and cost modeling work is useful for the estimation of execution times of CPC algorithms in general, and of agent-based epidemic simulation in particular, however, it does has some limitations. Below, we list some of the limitations that if addressed will further increase its usefulness.

- The current process requires the user to demermine the computational and communication components (number of classes) in their application. This is a limiting factor, because it may not be feasible for a user to determine the computational and communication components in his application.
- To get constants for the cost metric, the user has to go through a two step process: first collect the actual data about execution times of application, and second apply statistical regression methodology to extract the constants. The process may appear longer and as part of future work, it would be helpful to reduce the extraction effort.
- The cost metric is not applicable to multiple machine architecutres, so it would be useful to add this feature in the future.
- The cost metric doesn't have the capability to estimate the memory usage, and adding this feature will definitely add to the value of the metric.
- The metric could also again a lot of value by adding the capability for different network structures.

7.3 Future Research

In this dissertation, we have addressed the challenges faced when performing cost estimation of agent-based epidemic simulations and improving its scalability on high-performance clusters. Needless to say, there are still a number of open questions dealing with the efficient use of computing resources (high-performance clusters) for agent-based epidemic simulations. In the following section, I have outlined my vision, which is the natural extension of the work done in this dissertation.

7.3.1 Network-aware Cost Estimation Model

Agent-based epidemic simulations process large agent-location networks. The networks vary largely from each other in terms of their size and density. For example, in case of the US, the smallest network is a few thousand vertices (NRV social network), and the largest (US population social network) is more than 300 million vertices. Another issue is that the networks are different in terms of their density as well. For example, the density of rural social networks is smaller compared to the congested urban cities such as New York and Chicago.

The current cost estimator, although, is general in terms of disease dynamics and the size of the network (vertices only), it is unable to consider the densities of these networks. Therefore, it is natural to further investigate this aspect in detail to find a way to incorporate the same into the cost estimation equations. This can be done by adding another parameter to the cost equations that can quantify the network dynamics. This parameter can be network density, diameter, radius, or edge counts. It could also be a combination of these parameters.

7.3.2 Uncertainty Bounds for Random-Streams

Agent-based epidemic models are stochastic and require multiple replicates of each configuration to be run. Although, running multiple replicates helps achieve tighter bounds on the output, it makes the task of cost estimation challenging. We list this as part of our future work in order to study the effects of random streams on the compute time, and wish to provide the uncertainty bounds for all disease models. These uncertainty bounds will show the variation from using different random streams.

7.3.3 Machine Architecture-aware Cost Model

The constants of the current cost estimation model are very machine-specific and do not work in a similar fashion for another machine architecture. Therefore, it is necessary to extend the cost estimation model to generalize it across a variety of HPC clusters. Considering this, the cost metric can be modified by adding features such as processor speed, memory size, and network bandwidth.

7.3.4 Dynamic Load Balancing

Although, static load distribution is useful for assigning initial balanced workloads across processors, the load may not stay balanced for the duration of the simulation. In ABES, this is important as the simulations are iterative, caused by the changes in the state of agents, which in turn results in the changes in the workload. In fact, we have shown that the workloads do change across processors due to disease dynamics. To re-balance the changing workloads, we require dynamic load balancing. Further, the cost estimation metrics can be used to balance the load across processors. Therefore, due to the high cost associated with load migration, load balancing can be performed only every few iterations.

Bibliography

[1] M. E. Halloran, N. M. Ferguson, S. Eubank, I. M. Longini Jr, D. A. T. Cummings, B. Lewis, S. Xu, C. Fraser, A. Vullikanti, T. C. Germann, and et al. Modeling targeted layered containment of an influenza pandemic in the United States. *Proceedings of the National Academy of Sciences*, 105(12):4639, 2008.

[2] U.S. Department of Health and Human Services. *HHS Pandemic Influenza Plan*. U.S. Department of Health and Human Services, url: http://www.hhs.gov/pandemicflu/plan/, 2005.

[3] T. C. Germann, K. Kadau, I. M. Longini Jr, and C. A. Macken. Mitigation strategies for pandemic influenza in the United States. *Proceedings of the National Academy of Sciences*, 103(15):5935–5940, 2006.

[4] Committee on Modeling Community Containment for Pandemic Influenza. *Modeling Community Containment for Pandemic Influenza: A Letter Report*. National Academies Press, 2007.

[5] Christopher L. Barrett, Keith R. Bisset, Stephen G. Eubank, Xizhou Feng, and Madhav V. Marathe. Episimdemics: an efficient algorithm for simulating the spread of infectious disease over large realistic social networks. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[6] K.R. Bisset, A.M. Aji, E. Bohm, L.V. Kale, T. Kamal, M.V. Marathe, and Jae-Seung Yeom. Simulating the spread of infectious disease over large realistic social networks using Charm++. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 507 –518, may 2012.

[7] Keith R Bisset, Jiangzhuo Chen, Xizhou Feng, VS Kumar, and Madhav V Marathe. EpiFast: a fast algorithm for large scale realistic epidemic simulations on distributed memory systems. In *Proceedings of the 23rd international conference on Supercomputing*, pages 430–439. ACM, 2009.

[8] Keith R Bisset, Jiangzhuo Chen, Xizhou Feng, Yifei Ma, and Madhav V Marathe. Indemics: an interactive data intensive framework for high performance epidemic simulation. In *Proceedings of the 24th ACM International Conference on Supercomputing*, pages 233–242. ACM, 2010.

[9] Keith Bisset, Xizhou Feng, Madah Marathe, and Shrirang Yardi. Modeling interaction between individuals, social networks and public policy to support public health epidemiology. In *Proceedings of the 2009 Winter Simulation Conference*, pages 2020–2031, December 2009.

[10] S. Eubank. Scalable, efficient epidemiological simulation. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 139–145, New York, NY, USA, 2002. ACM.

[11] S. Eubank, H. Guclu, M. V. Marathe, and et al. Modelling disease outbreaks in realistic urban social networks. *Nature*, 429(6988):180–184, May 2004.

[12] I. Longini, A. Nizam, et al. Containing pandemic influenza at the source. *Science*, 309(5737):1083–1087, 2005.

[13] N. M. Ferguson, M. J. Keeling, and et al. Planning for smallpox outbreaks. *Nature*, 425(6959):681–685, 2003.

[14] J. Parker. A flexiable, large-scale, distributed agent based epidemic model. In *Winter Simulation Conference*, 2007.

[15] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*, volume 12, page 2, 2012.

[16] Amine Abou-Rjeili and George Karypis. Multilevel algorithms for partitioning power-law graphs. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 10–pp. IEEE, 2006.

[17] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[18] RM Karp and N Karmarker. The differencing method of set partitioning. *UCB/CSD Report*, 1982.

[19] Chien-Chung Shen and Wen-Hsiang Tsai. A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. *Computers, IEEE Transactions on*, C-34(3):197 –203, march 1985.

[20] Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European journal of operational research*, 197(2):427–438, 2009.

[21] George Karypis and Vipin Kumar. Metis - unstructured graph partitioning and sparse matrix ordering system, version 2.0. Technical report, University of Minniesota, USA, 1995.

[22] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Supercomputing, 1996. Proceedings of the 1996 ACM/IEEE Conference on*, pages 35–35. IEEE, 1996.

[23] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[24] François Pellegrini and Jean Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *High-Performance Computing and Networking*, pages 493–498. Springer, 1996.

[25] Chung-Lun Li and TCE Cheng. The parallel machine min-max weighted absolute lateness scheduling problem. *Naval Research Logistics (NRL)*, 41(1):33–46, 1994.

[26] Chris HQ Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst D Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 107–114. IEEE, 2001.

[27] F. Ercal, J. Ramanujam, and P. Sadayappan. Task allocation onto a hypercube by recursive mincut bipartitioning. *Journal of Parallel and Distributed Computing*, 10(1):35 – 44, 1990.

[28] P Sadayappan and Fikret Ercal. Cluster-partitioning approaches to mapping parallel programs onto a hypercube. In *Supercomputing*, pages 475–497. Springer, 1988.

[29] Nikhil Bansal, Uriel Feige, Robert Krauthgamer, Konstantin Makarychev, Viswanath Nagarajan, Joseph Seffi, and Roy Schwartz. Min-max graph partitioning and small set expansion. *SIAM Journal on Computing*, 43(2):872–904, 2014.

[30] Angelia Nedic and Asuman Ozdaglar. Distributed subgradient methods for multi-agent optimization. *Automatic Control, IEEE Transactions on*, 54(1):48–61, 2009.

[31] Concepció Roig, Ana Ripoll, Miquel A Senar, Fernando Guirado, and Emilio Luque. Modelling message-passing programs for static mapping. In *Parallel and Distributed Processing, 2000. Proceedings. 8th Euromicro Workshop on*, pages 229–236. IEEE, 2000.

[32] Evelyn Duesterwald and Vasanth Bala. Software profiling for hot path prediction: less is more. *ACM SIGARCH Computer Architecture News*, 28(5):202–211, 2000.

[33] Donald Alpert and Gary N Hammond. Method and apparatus for providing breakpoints on taken jumps and for providing software profiling in a computer system, August 19 1997. US Patent 5,659,679.

[34] Rajendra Patel and Arvind Rajwat. A survey of embedded software profiling methodologies. *arXiv preprint arXiv:1312.2949*, 2013.

[35] Harold S Stone and Shahid H Bokhari. Control of distributed processes. *Computer*, 11(7):97–106, 1978.

[36] Harold S Stone. Multiprocessor scheduling with the aid of network flow algorithms. *Software Engineering, IEEE Transactions on*, 1(1):85–93, 1977.

[37] Yuan-Chieh Chow and Walter H Kohler. Models for dynamic load balancing in a heterogeneous multiple processor system. *Computers, IEEE Transactions on*, 100(5):354–361, 1979.

[38] Wesley W Chu, Leslie J Holloway, Min-Tsung Lan, and Kemal Efe. Task allocation in distributed data processing. *Computer*, 13(11):57–69, 1980.

[39] Perng-Yi Richard Ma, Edward Y. S. Lee, and Masahiro Tsuchiya. A task allocation model for distributed computing systems. *IEEE Transactions on Computers*, 31(1):41–47, 1982.

[40] Virginia Mary Lo. Heuristic algorithms for task assignment in distributed systems. *Computers, IEEE Transactions on*, 37(11):1384–1397, 1988.

[41] H. Renard, Y. Robert, and F. Vivien. Static load-balancing techniques for iterative computations on heterogeneous clusters. *Euro-Par 2003 Parallel Processing*, pages 148–159, 2003.

[42] B. Cosenza, G. Cordasco, R. De Chiara, and V. Scarano. Distributed load balancing for parallel agent-based simulations. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, pages 62–69. IEEE, 2011.

[43] U.V. Catalyurek, E.G. Boman, K.D. Devine, D. Bozdag, R. Heaphy, and L.A. Riesen. Hypergraph-based dynamic load balancing for adaptive scientific computations. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1 –11, march 2007.

[44] Sunil Chopra and MR Rao. The partition problem. *Mathematical Programming*, 59(1-3):87–115, 1993.

[45] Sunil Chopra and MR Rao. Facets of the k-partition polytope. *Discrete Applied Mathematics*, 61(1):27–48, 1995.

[46] Ulas Karaoz, TM Murali, Stan Letovsky, Yu Zheng, Chunming Ding, Charles R Cantor, and Simon Kasif. Whole-genome annotation by using evidence integration in functional-linkage networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101(9):2888–2893, 2004.

[47] Damon Centola and Michael Macy. Complex contagions and the weakness of long ties1. *American Journal of Sociology*, 113(3):702–734, 2007.

[48] Damon Centola. The spread of behavior in an online social network experiment. *science*, 329(5996):1194–1197, 2010.

[49] Brandon G Aaby, Kalyan S Perumalla, and Sudip K Seal. Efficient simulation of agent-based models on multi-gpu and multi-core clusters. In *proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, page 29. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.

[50] Richard D Smith. Responding to global infectious disease outbreaks: Lessons from sars on the role of risk perception, communication and management. *Social science & medicine*, 63(12):3113–3123, 2006.

[51] Walter F Stewart, Judith A Ricci, Elsbeth Chee, and David Morganstein. Lost productive work time costs from health conditions in the united states: results from the american productivity audit. *Journal of Occupational and Environmental Medicine*, 45(12):1234–1246, 2003.

[52] M Zia Sadique, W John Edmunds, Richard D Smith, William Jan Meerding, Onno De Zwart, Johannes Brug, and Philippe Beutels. Precautionary behavior in response to perceived threat of pandemic influenza. *Emerging infectious diseases*, 13(9):1307, 2007.

[53] Elaine O Nsoesie, Richard J Beckman, and Madhav V Marathe. Sensitivity analysis of an individual-based model for simulation of influenza epidemics. *PloS one*, 7(10):e45414, 2012.

[54] Elaine Nsoesie, Madhav Mararthe, and John Brownstein. Forecasting peaks of seasonal influenza epidemics. *PLoS currents*, 5, 2013.

[55] Stephen Eubank, Hasan Guclu, VS Anil Kumar, Madhav V Marathe, Aravind Srinivasan, Zoltan Toroczkai, and Nan Wang. Modelling disease outbreaks in realistic urban social networks. *Nature*, 429(6988):180–184, 2004.

[56] Chris Barrett, Keith Bisset, Jonathan Leidig, Achla Marathe, and Madhav Marathe. Estimating the impact of public and private strategies for controlling an epidemic: A multi-agent approach. In *Proceedings of the 21st IAAI Conference*, 2009.

[57] Jae-Seung Yeom, Abhinav Bhatele, Keith Bisset, Eric Bohm, Abhishek Gupta, Laxmikant V Kale, Madhav Marathe, Dimitrios S Nikolopoulos, Martin Schulz, and Lukasz Wesolowski. Overcoming the scalability challenges of epidemic simulations on blue waters. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 755–764. IEEE, 2014.

[58] Laxmikant V Kale and Gengbin Zheng. Charm++ and Ampi: Adaptive runtime strategies via migratable objects. *Advanced Computational Infrastructures for Parallel and Distributed Applications*, pages 265–282, 2009.

[59] Keith R. Bisset, Ashwin M. Aji, Tariq Kamal, Jae-Seung Yeom, Madhav V. Marathe, Eric J. Bohm, and Abhishek Gupta. Contagion diffusion with EpiSimdemics. In Laxmikant V. Kale and Abhinav Bhatele, editors, *Parallel Science and Engineering Applications: The Charm++ Approach*, pages 207–242. Taylor & Francis Group, CRC Press, 2014.

[60] Christopher L Barrett, Keith Bisset, Stephen Eubank, Madhav V Marathe, VS Anil Kumar, and Henning S Mortveit. Modeling and simulation of large biological, information and socio-technical systems: An interaction-based approach. In *Proceedings of Symposia in Applied Mathematics*, volume 64, page 101, 2007.

[61] A Eliot Shearer, Adam P DeLuca, Michael S Hildebrand, Kyle R Taylor, José Gurrola, Steve Scherer, Todd E Scheetz, and Richard JH Smith. Comprehensive genetic testing for hereditary hearing loss using massively parallel sequencing. *Proceedings of the National Academy of Sciences*, 107(49):21104–21109, 2010.

[62] Rammohan Mallipeddi, Ponnuthurai N Suganthan, Quan-Ke Pan, and Mehmet Fatih Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied Soft Computing*, 11(2):1679–1696, 2011.

[63] John J Grefenstette, David E Moriarty, and Alan C Schultz. Evolutionary algorithms for reinforcement learning. *arXiv preprint arXiv:1106.0221*, 2011.

[64] Javier J Sánchez-Medina, Manuel J Galán-Moreno, and Enrique Rubio-Royo. Traffic signal optimization in la almozara district in saragossa under congestion conditions, using genetic algorithms, traffic microsimulation, and cluster computing. *Intelligent Transportation Systems, IEEE Transactions on*, 11(1):132–141, 2010.

[65] Ammar Al-Bazi and Nashwan Dawood. Developing crew allocation system for the precast industry using genetic algorithms. *Computer-Aided Civil and Infrastructure Engineering*, 25(8):581–595, 2010.

[66] Shanfei Jiao, Chen He, Yusheng Dou, and Hong Tang. Molecular dynamics simulation: Implementation and optimization based on hadoop. In *Natural Computation (ICNC), 2012 Eighth International Conference on*, pages 1203–1207. IEEE, 2012.

[67] Yasset Perez-Riverol, Roberto Vera, Yuliet Mazola, and Alexis Musacchio. A parallel systematic-monte carlo algorithm for exploring conformational space. *Current topics in medicinal chemistry*, 12(16):1790–1796, 2012.

[68] Ira M Longini, Azhar Nizam, Shufu Xu, Kumnuan Ungchusak, Wanna Hanshaoworakul, Derek AT Cummings, and M Elizabeth Halloran. Containing pandemic influenza at the source. *Science*, 309(5737):1083–1087, 2005.

[69] Jon Parker and Joshua M Epstein. A distributed platform for global-scale agent-based models of disease transmission. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 22(1):2, 2011.

[70] Kalyan S Perumalla and Sudip K Seal. Discrete event modeling and massively parallel execution of epidemic outbreak phenomena. *Simulation*, 88(7):768–783, 2012.

[71] Karthik Channakeshava, Keith Bisset, VS Anil Kumar, Madhav Marathe, and Shrirang Yardi. High performance scalable and expressive modeling environment to study mobile malware in large dynamic networks. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 770–781. IEEE, 2011.

[72] Keith Bisset, Md Alam, Josep Bassaganya-Riera, Adria Carbo, Stephen Eubank, Raquel Hontecillas, Stefan Hoops, Yongguo Mei, Katherine Wendelsdorf, Dawen Xie, et al. High-performance interaction-based simulation of gut immunopathologies with enteric immunity simulator (ENISI). In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 48–59. IEEE, 2012.

[73] Yongguo Mei, Raquel Hontecillas, Xiaoying Zhang, Keith Bisset, Stephen Eubank, Stefan Hoops, Madhav Marathe, and Josep Bassaganya-Riera. ENISI Visual, an agent-based simulator for modeling gut immunity. In *Bioinformatics and Biomedicine (BIBM), 2012 IEEE International Conference on*, pages 1–5. IEEE, 2012.

[74] Charles M Macal and Michael J North. Agent-based modeling and simulation. In *Winter Simulation Conference*, pages 86–98. Winter Simulation Conference, 2009.

[75] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[76] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 810–818. ACM, 2010.

[77] Jaeseok Myung, Jongheum Yeon, and Sang-goo Lee. Sparql basic graph pattern processing with iterative mapreduce. In *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud*, page 6. ACM, 2010.

[78] David E Goldberg and John H Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[79] David Edward Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms by David E. Goldberg*, volume 7. Springer, 2002.

[80] David E Goldberg, Bradley Korb, and Kalyanmoy Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex systems*, 3(5):493–530, 1989.

[81] Gilbert Syswerda. Uniform crossover in genetic algorithms. *Proceedings of the third international conference on Genetic Algorithms*, 1989.

[82] Joshua M Epstein, Jon Parker, Derek Cummings, and Ross A Hammond. Coupled contagion dynamics of fear and disease: mathematical and computational explorations. *PLoS One*, 3(12):e3955, 2008.

[83] Athanasios C Antoulas. *Approximation of large-scale dynamical systems*, volume 6. Siam, 2005.

[84] Eugene Asarin, Olivier Bournez, Thao Dang, and Oded Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In *Hybrid Systems: Computation and Control*, pages 20–31. Springer, 2000.

[85] Hung V Ly and Hien T Tran. Modeling and control of physical processes using proper orthogonal decomposition. *Mathematical and computer modelling*, 33(1):223–236, 2001.

[86] Norman Richard Draper, Harry Smith, and Elizabeth Pownell. *Applied regression analysis*, volume 3. Wiley New York, 1966.

[87] Phillip I Good and James W Hardin. *Common errors in statistics (and how to avoid them)*. Wiley, 2012.

[88] George Karypis and Vipin Kumar. Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 48(1):96–129, 1998.

[89] Tariq Kamal, Keith R Bisset, Ali R Butt, Youngyun Chungbaek, and Madhav Marathe. Load balancing in large-scale epidemiological simulations. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 123–124. ACM, 2013.

[90] SPSS statistics for windows, IBM Corp.

[91] Gene M. Amdahl. Validity of the single-processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485, 1967.

[92] Sandeep Sharma, Sarabjit Singh, and Meenakshi Sharma. Performance analysis of load balancing algorithms. *World Academy of Science, Engineering and Technology*, 38:269–272, 2008.

[93] Haakon Bryhni, Espen Klovning, and Øivind Kure. A comparison of load balancing techniques for scalable web servers. *Network, IEEE*, 14(4):58–64, 2000.

[94] George Karypis, Kirk Schloegel, and Vipin Kumar. Parmetis: Parallel graph partitioning and sparse matrix ordering library. *Version 1.0, Dept. of Computer Science, University of Minnesota*, 1997.

[95] Timothy C Reluga, Jan Medlock, and Alan S Perelson. Backward bifurcations and multiple equilibria in epidemic models with structured immunity. *Journal of theoretical biology*, 252(1):155–165, 2008.

[96] Matt J Keeling and Pejman Rohani. *Modeling infectious diseases in humans and animals*. Princeton University Press, 2008.

[97] Horst R Thieme. Persistence under relaxed point-dissipativity (with application to an endemic model). *SIAM Journal on Mathematical Analysis*, 24(2):407–435, 1993.

[98] Lukasz Wesolowski, Ramprasad Venkataraman, Arpan Gupta, Jae-Seung Yeom, Keith Bisset, Yanhua Sun, Pritish Jetley, Thomas R Quinn, and Laxmikant V Kalé. Tram: Optimizing fine-grained communication with topological routing and aggregation of messages. In *Parallel Processing (ICPP), 2014 43rd International Conference on*, pages 211–220. IEEE, 2014.

[99] Christopher L Barrett, Richard J Beckman, Maleq Khan, VS Anil Kumar, Madhav V Marathe, Paula E Stretz, Tridib Dutta, and Bryan Lewis. Generation and analysis of large synthetic social contact networks. In *Winter Simulation Conference*, pages 1003–1014. Winter Simulation Conference, 2009.

[100] Keith Bisset, Jiangzhuo Chen, Chris J Kuhlman, VS Kumar, and Madhav V Marathe. Interaction-based hpc modeling of social, biological, and economic contagions over large networks. In *Proceedings of the winter simulation conference*, pages 2938–2952. Winter Simulation Conference, 2011.

[101] Sudip Saha, Abhijin Adiga, and Anil Kumar S Vullikanti. Equilibria in epidemic containment games. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[102] Jiangzhuo Chen, Fei Huang, Maleq Khan, Madhav Marathe, Paula Stretz, and Huadong Xia. The effect of demographic and spatial variability on epidemics: A comparison between beijing, delhi, and los angeles. In *Critical Infrastructure (CRIS), 2010 5th International Conference on*, pages 1–8. Citeseer, 2010.

[103] Claudia Taylor, Achla Marathe, and Richard Beckman. Same influenza vaccination strategies but different outcomes across us cities? *International Journal of Infectious Diseases*, 14(9):e792–e795, 2010.