Modeling and Computation of Complex Interventions to Control Disease Outbreak in Large-scale Epidemiological Simulations using SQL and Distributed Database

Rushi Kaw

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

Master of Science
in
Computer Science

Madhav V. Marathe, Chair
Sandeep Gupta
B. Aditya Prakash

July 24, 2014
Blacksburg, Virginia

Keywords: epidemic simulation, distributed system, database system

Modeling and Computation of Complex Interventions to Control Disease Outbreak in Large-scale Epidemiological Simulations using SQL and Distributed Database

Rushi Kaw

(ABSTRACT)

Scalability is an important problem in epidemiological applications that simulate complex intervention scenarios over large datasets. Indemics is one such interactive data intensive framework for High-performance computing(HPC) based large-scale epidemic simulations. In the Indemics framework, interventions are supplied from an external, standalone database which proved to be an effective way of implementing interventions. Although this setup performs well for simple interventions and small datasets, performance and scalability of complex interventions and large datasets remain an issue. In this thesis, we present IndemicsXC, a scalable and massively parallel high-performance data engine for Indemics in a supercomputing environment. IndemicsXC has the ability to implement complex interventions over large datasets. Our distributed database solution retains the simplicity of Indemics by using the same SQL query interface for expressing interventions. We show that our solution implements the most complex interventions by intelligently offloading them to the supercomputer nodes and processing them in parallel. We present an extensive performance evaluation of our database engine with the help of various intervention case studies over synthetic population datasets. The evaluation of our parallel and distributed database framework illustrates its scalability over standalone database. Our results show that the distributed data engine is efficient as it is parallel, scalable and cost-efficient means of implementing interventions. The proposed cost-model in this thesis could be used to approximate intervention query execution time with decent accuracy. The usefulness of our distributed database framework could be leveraged for fast, accurate and sensible decisions by the public health officials during an outbreak. Finally, we discuss the considerations for using distributed databases for driving large-scale simulations.

Dedication

I dedicate this work to my grandfather, parents and to my sister.

Acknowledgments

Foremost, I would like to thank my committee members for their valuable guidance and support. In particular, I owe my deepest gratitude to my adviser, Dr. Sandeep Gupta for being a great mentor. This work has been made possible because of his valuable inputs and guidance. Working with him helped me discover my areas of improvement. Dr. Sandeep gave me the confidence to move forward and has been ever so supportive and encouraging in my work. I learned a lot from him and would like to thank him for sharing his knowledge with me.

I am grateful to Dr. Madhav Marathe for giving me the opportunity to work with him. I am thankful for his insightful comments about my work that helped shape my thesis and gave it the right direction. I would also like to thank Dr. Aditya Prakash for his valuable inputs in my thesis.

I also thankful to my colleagues at NDSSL for being so kind and inspiring me in so many ways. In the end, I would like to thank all my friends at Virginia tech and my family for their love, encouragement and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Due to the constant threat of a pandemic, public health officials and policy makers are busy making decisions which affect the human society. The 2009 H1N1 outbreak illustrates one of the situations where systems instituted through pandemic planning were used for the prevention and control of the pandemic. The current outbreak of the deadly Ebola virus in West Africa has prompted health officials to take containment measures to prevent its spread outside Africa [42]. The unpredictability of such disease outbreaks entails global preparedness and development of effective means for preparing against the next pandemics onslaught. These situations necessitate collaboration between the computational scientists and the epidemiologists so that the disease and intervention simulations would help determine the spread of a pandemic. As commented by UN System Influenza Coordinator David Nabarro, 2013:

> Preparedness, including for potential pandemics, requires coordination and management of complex relationships across different sectors and between international, national, and local actors. We must work together in support of all societies as they prepare in ways that reflect the interests of all people for whom preparations are being made.

The amalgamation of network science and epidemiology has revolutionized the research into infectious disease dynamics. Propagation of a disease in a contact network through connections between individuals has been studied to gain insights into the infection growth and design disease control strategies. Hence, contact network epidemiology plays a significant role in understanding the disease spread in a network and informing disease control. Various mathematical and computational models have been built by organizations [5, 8, 40] to study the disease dynamics on different contact networks. Some of these include differential equation based models [21], agent-based models, and the computational models [5, 7]. In

comparison to the models based on a differential equation, the computational models introduce high-fidelity in the simulation by describing the contagion diffusion across a social contact network generated by a synthetic population generator [4].

A typical workflow for predicting the outcome of contagion (or epidemic) is as follows: First, a disease spread model such as SEIR, SIR or a probability function is decided. Then a synthetic population generator is used for creating a realistic demographic. That is the resulting demographic would match the actual population base as closely as possible on parameters relevant for contagion study such as type and geographical location of houses, the inhabitants of these houses, their age, gender, income and most importantly, their daily activities. Third, a computational algorithm that uses the disease model to simulate the contagion-diffusion over the given synthetic information. The computation takes as input an instance of the disease model and population data from the synthetic generator. The computational engine performs agent based discrete time step simulation. The duration gets divided into a fixed number of time steps. In each time step, the health state of each individual is computed based on the disease model and the activity recorded by the synthetic generator. It makes the results from such high-fidelity models far more robust and accurate as they capture the Spatio-temporal aspects of contact and the spread of disease. These models, however, do not support real-time assessment of ongoing simulation that is a desirable feature as it helps the policy makers to take informed decisions in real time. The need for real-time simulation result analysis has been discussed in major epidemiology articles [14] and elsewhere [23, 27]. In a recent work [12], researchers have used tweets about health topics from social-media to learn about the public health and possible disease outbreaks. Also, in these computational models, the intervention scenarios are implemented inside the simulation engine, which hinders modularity and usability of such tools.

Motivated by such events, Indemics [8] was developed. It is an interactive, high performance modeling framework for real-time pandemic planning. EpiFast is the computational model used in the Indemics setup. In Indemics, the contagion diffusion takes place on an HPC system while the interventions are supplied from an external Database Management System(DBMS). In contrast to EpiFast, Indemics supports interventions computed and supplied from an external standalone DBMS. Interventions within a contagion simulation refer to altering the behavior and state of agent based on the current state of the simulation. In Indemics, interventions are realized in the form of database queries. The parameters provided by policy makers are translated into queries in a database and processed in the HPC simulation engine. For example, in school study, the decision to intervene a school population with 50% compliance would result in half of the school population being intervened on the first day. Indemics improves the usability, modularity and extensibility by providing an efficient way of implementing the interventions, reducing the effort to add new interventions and simplify the design of the simulation engine.

1.1.1 Interventions, Cells, and Replicates: A complex and read-/write interleaved data-intensive workload

Interventions within a contagion-diffusion simulation refer to altering the behavior and state of agent-based on the current state of the simulation. The type of intervention that can be applied varies with respect to the experiment setup and the demographic, social, and, environmental attributes modeled by agents in the simulation. These different types of interventions are realized in terms of database queries. Compared to other aspects of simulation, the data manipulation involved is highly complex and irregular. Even simple intervention rule such as "close school in blocks where fraction of child infection has crossed a threshold" requires sieving, filtering, aggregating, and collating six to seven different collection of datasets (the close school example requires school location, children home location, home location a in block or tract, and daily state of individuals). If the demography being studied is large such as state of Texas or entire U.S. population, this operation itself is highly sophisticated and resource and time intensive. Realistic interventions involves a combination of such or more intertwined rules. Typically these include social distancing, combined with antivirals and vaccinations, and, policy decisions (close down school, reduce transport etc.).

Applying such interventions entail making updates to the existing data. These typically include changing the health status (susceptible, infected, exposed, recovering) of the individuals or a demographic location such as a block. Simulations in other domains do not have updates i.e. change value of existing data item. Typically they run in phases, each phase either performing bulk read (the input) or bulk writes (the output). On the contrary, updates are not only required for interventions, which are the order of magnitudes complex then bulk write operations but they are also intertwined with the read operations.

It turns out that this agent-based approach along with interventions is far effective than the mathematical models because they describe the spread of contagion in a contact network. However, computing the simulation is both compute and data intensive. It would be impractical to optimize each class of interventions as there could potentially be infinitely many combinations. Treating the problem in its generality is hard because of two reasons: first, we need to define a simple way to formulate all types of interventions, and second, given the formulation and simulation data, somehow apply the intervention rules efficiently. The intervention rules vary and have unpredictable manipulation operations and perform updates interleaved with reads (unlike bulk reads followed by bulk writes). That makes it a non-trivial problem that has not been studied outside of computational epidemiology domain.

Cells and Replicates: Due to the factorial experiment design of the epidemiology studies and the probabilistic disease propagation in simulations, the computational requirements grow exponentially. An intervention study is a collection of simulations on the same base population but with different input parameters. For e.g. a study that runs a simulation for each combination efficacy (low, medium, and, high) and (.1%, 1%, 2%, 5%) vaccination values. A typical simulation has six to ten such domain-specific parameters each of which

typically has eight to ten distinct values. A combination of values of parameters is called a "cell". A cell maps to configuration of a single experiment. Results from multiple simulations for the same cell (or configuration) but with different random seeds (these are called replicates) are used to obtain stochastically robust results. The need for high fidelity for more realistic and reliable results, incorporating complex interventions, factorial (or partial factorial) experiment setup for discovering effective strategies and multiple runs for statistically robust results lead to massive scale database operations, compute processing and bandwidth requirements.

1.2 Motivation

In Indemics, state-of-art contagion diffusion code can be scaled by using high-performance computing approaches and adding processing resources [44]. However, in these simulations, bulk of time is consumed by the database queries when computing interventions. Although databases prove to be effective in supporting interventions in large scale simulations, scalability remains an issue. The problem here is that with a lot of data to process, standalone databases don't seem to handle the load well. The database performance further degrades as the query complexity increases when modeling more realistic interventions scenarios. To maintain high productivity of Indemics, we need to enhance the productivity of databases as well. To that end, we introduce a distributed and massively parallel data engine, integrated with the present Indemics framework for scaling the interventions. By leveraging the parallelism and scalability offered by a distributed database, interventions queries can be processed in parallel, enabling fast simulation and thus making real-time pandemic planning more efficient. Many applications that involve massive amounts of computations make use of supercomputing environment for running large-scale simulations. The supercomputers provide a powerful computing environment for solving complex problems. They have a ubiquitous presence everywhere, from weather forecasting to running large-scale epidemiological simulations. Because of the resources that an HPC system offers, they become an obvious choice for supporting a distributed database. These systems sustain a conducive environment for tasks with data processing needs. Hence, in our proposed framework, simulation engine and the data engine reside in the HPC environment.

1.3 Premise of the work

In this work, we propose the following:

1. For the class of intervention queries studied in this thesis, our distributed database setup performs better than the current standalone DBMS.

2. The distributed data-engine scales as a function of social contact network size. Our

performance evaluation encompasses both strong and weak scaling by varying interventions and distributed database nodes. A fixed dataset should scale as we add more distributed datanodes.

3. When multiple simulation cells run in parallel, selecting the right work/data distribution strategy yields maximum throughput. The strategy varies as per the contact network size and the intervention scenario.

4. Applying the optimization strategies, to be discussed in Chapter 7, reduces the running time of the intervention queries studied in this work.

1.4 Contributions

Here, we present IndemicsXC, a scalable and distributed database solution for driving complex and data-intensive intervention scenarios in contact network epidemiology. Our work builds upon the existing Indemics framework with the main focus on optimizing the database component of the HPC-based simulation environment. The current Indemics framework is not able to handle complex and data-intensive intervention scenarios. This work addresses these shortcomings of the standalone database-based Indemics model. The main objective of this thesis is to make the database operations for intervention computation as efficient as possible. We achieve this by replacing the standalone DBMS used in the previous framework by a high-performance distributed database and yet retaining the simplicity of using the same SQL query interface for expressing interventions. In the current Indemics framework, the database component resides outside of the HPC environment, on a standalone machine. In contrast, the distributed database in our framework resides inside the HPC environment with the computational simulation engine. Our evaluation and results suggest that the distributed DBMS with massively parallel query processing is a more promising choice for supporting interventions in a simulation than a standalone database. Our main contributions can be summarized as below:

- **A New Approach:**
  We introduce a novel approach of using distributed database system on HPC clusters for implementing complex intervention scenarios. Our approach offers a scalable solution for driving complex interventions, can handle multiple instances of the simulation and avoid massive data movement in and out of the supercomputer. Utilizing supercomputers for massive scale data and compute-intensive application has been one of the recent goals for the HPC community [26]. Although there has been an enormous growth in the use of parallel and distributed databases in large scale simulations [22, 43], this is the first time they are being used for supporting the realistic intervention scenarios. The read/write scalability, parallel data processing and ability to handle data-intensive queries makes them an ideal choice for supporting interventions.

- **Query optimizations:**
  We apply and implement various query optimization strategies for accelerating the processing of the intervention queries. We try to solve two problems here - first, improving the performance of a single instance of simulation and second, improving the throughput performance of multiple instances of a simulation running in parallel. Hence, we study techniques for scaling a single cell and scaling throughput when running multiple cells. We present strategies based on data distribution schemes, incorporating simulation-specific changes in the optimizer, using partitioning schemes on a single node and different ways of mapping simulation cells to distributed worker nodes. We also present heuristic rules for optimal data distribution. In addition to the optimization strategies, we also tune various database parameters that are critical in improving the query performance. Some of these parameters, such as the ones that decide whether a query should go to disk or not, can hugely impact a query. Most of these parameters have been tweaked as per our system for general performance improvement.

- **Evaluation:**
  We report an extensive evaluation of our distributed database-based IndemicsXC framework based on various intervention case studies. These selected case studies simulate different types of workloads (in terms of read/ write and SQL operations) which helps in evaluating the effectiveness and efficiency of our framework and the optimization techniques. We present the results for case-studies that were both feasible and not feasible on the Indemics framework. We also present a comparative analysis of Indemics and IndemicsXC for driving selected intervention scenarios in networked epidemiology.

- **Cost Model:**
  We propose a cost model for estimating the cost of a given intervention query. The model comprises a set of cost functions that assign costs to atomic query operations. It also takes into account the costs associated with a distributed database system such as data transfer time and the aggregation time at the master node. Based on the query cost, the model computes the amount of resources (number of datanodes) to be allocated to a given simulation experiment. We use experimental results from various intervention studies to validate the accuracy of our cost model.

Through the above contributions, we have tried to prove the premises listed in Section 1.3. From different studies and experimentations, we conclude the following:

- **Premise 1**: The performance results described in section 8.1.1 illustrate that our distributed database framework performs better than the current standalone DBMS. We observe query processing speedup in all the studies simulated in this work. We get as much as 20**X** performance improvement in the distance-1 study which is considered to be one of the complex intervention scenarios.

- **Premise 2**: Our distributed database scales easily for the realistic studies over contact networks of varying sizes. The results in Section 8.2 show that the distributed system

scales well to large networks with millions of agents. We observe both weak and strong scaling in all the interventions. The speedup is much more prominent in interventions with complex query operations, more so in large and dense datasets. We observe a strong scaling of up to 7**X** in LA distance-1 run.

- **Premise 3**: For multiple simulation instances running in parallel, two task/data distribution methods are proposed: all-to-all and one-to-one. These methods were proposed to reduce the execution time per instance, hence improving the overall throughput of the simulations. Our results (refer Section 8.3) show that for a given intervention queries and contact networks, all-to-all performs better than one-to-one. We observe that all-to-all performs twice as better as one-to-one for Chicago block intervention.

- **Premise 4**: From the performance results shown in Section 8.4 we establish the effectiveness of the various optimization techniques described in Chapter 7. For instance, we observe about 12**X** improvement in Miami block intervention by making the changes in the query optimizer such that the Inserts(predominant operation in almost all intervention scenarios) get pushed down. We observe similar speedups(about 22**X**) by pushing down Update operation.

Chapter 2

Indemics Framework Coupled with Database-driven Simulation

2.1 Indemics Framework

In the previous section, we briefly introduced the Indemics model with emphasis on decoupling of the compute-intensive and data-intensive tasks. We now present an architectural implementation of the Indemics framework and its components (see Figure 2.1). Indemics consists of four loosely coupled modules: Indemics Middleware platform, Indemics Epidemic simulation engine, Indemics Data-engine for intervention simulation and an Indemics Client for supplying the interventions.

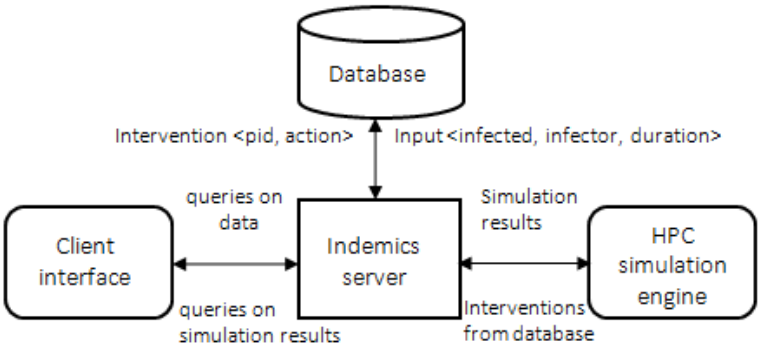


Figure 2.1: Overview of the Indemics framework. Indemics server is at the heart of the architecture which synchronizes the communication between rest of the components viz. simulation engine, relational database and the Indemics client.

Middleware platform is at the center of this framework and is responsible for synchronizing the interaction between the simulation engine, client and the data-engine. In the current Indemics implementation, Indemics server forms the middleware component. The Indemics

server module runs as a background service or daemon and keeps waiting for the simulation and client connections. Once the server receives a connection from the simulation, it sets up the computation. It synchronizes the execution of EpiFast and data movement to and from data engine for each time step of the simulation. EpiFast [7] act as a simulation engine, configured on an HPC based platform. Data engine in Indemics is a DMBS, which stores the simulation tables and also supports the intervention simulation. The Client is an SQL interface to the Indemics system. The current Client implementation provides an interactive console interface as well as a batch client script. In this body of work, we have used the batch client interface, which uses a script file consisting of a set of intervention rules.

Indemics server retrieves the current health state (whether in healthy, infected or intervened state) from the data engine (by executing a SQL query through the interface) and updates the EpiFast internal data structure accordingly. The server then resumes the EpiFast simulation and runs the model for discrete time step. The new states are then read from the internal data structures of the EpiFast and sent to the data engine (again using the SQL query). Next, it applies the interventions which amount to a collection of join, group-by, and, update in the database. We discuss the database setup and the interventions in more detail in Chapter 6.

The database in the current Indemics framework is a standalone DBMS and resides on a high-end server platform. In our proposed framework, the standalone We replace the standalone DBMS with a distributed and massively parallel data engine which resides within the HPC environment. Hence, in contrast to the earlier framework, both EpiFast and data engine in the new scheme reside within the same supercomputer. Direct advantage of using supercomputing resources are acceleration of epidemiology simulations and increase in throughput of intervention studies

This work emphasizes on the parallel and distributed processing of the underlying data processing engine of Indemics [8], which assumes relational interface for data persistence and computation over the data engine.

2.2 Coupled High Performance Diffusion and Interventions

The problem of scalable contagion diffusion has recently been an active area of research in computational epidemiology. In EpiFast [7], the authors addressed the problem of parallel computation of an epidemiology simulation. It studies the spread of contagion on a person-person contact network. EpiSimdemics [5] is a high performance implementation of contagion diffusion over person-location network. In the graph representation, a person is represented by a node and an edge denotes an activity of a person to a location. An interaction said to happen if two people come in contact at a location. The synthetic model in EpiSimdemics is more realistic as it captures the spatiotemporal aspects of contact and the spread of disease.

In such simulations, interventions are realized by changing the edge and node properties in the network. In EpiFast, the interventions were coded completely within the simulation engine. However, the scalability issues with this approach motivated the development of Indemics platform [8] where an external database management system was used to compute the interventions. Their work was a novel approach to couple a relational database with a high-performance simulation framework. This use of database addresses the problem of data management and heterogeneity in intervention across simulations. Initially, all the states of the simulation and inputs are maintained in the databases. Contagion diffusion is then coupled with the database where the interventions are expressed in a relational query language (SQL) [8]. The SQL is declarative, i.e., the user expresses the desired results but not the actual operation. The database engine, based on the expression, comes up with an efficient strategy to drive the intervention queries. It allows us to be able to express wide class of intervention rules without resorting to implementing each one of them which would have been a time consuming and costly endeavor. However, the other aspects of complex read and interleaved update operations limit the scalability.

The primary reason for this limitation is that databases are designed for enterprise and banking workloads where concurrent access (i.e., multiple clients accessing the same data) and consistency and recoverability of data is of utmost importance. Therefore, databases, typically tradeoff speed of writes and updates for reliability. A single update happens in multiple phases (update catalog, logging, checkpoint, triggers, indexing, disk page), and each phase has several synchronization operations such as locks. It is well known that these operations are expensive in the current system. All this leads to limited scalability. Second, the traditional or standalone databases do not make use of the hardware and parallelism available on supercomputers.

In the Indemics framework with a standalone DBMS, the simulation runs on the supercomputer while the database is outside the HPC environment, on a local area network. Although this framework offers flexibility and demonstrates the feasibility of running a simulation from a database, it did not meet the resource demands of running factorial design based simulation study in a timely manner. The data movement to-and-fro turns out to be quite expensive. Due to the standalone nature of database, complex demographic based interventions cannot be studied. For instance, a fairly complex intervention like distance-1, in which health state of one-hop neighbor needs to be calculated, takes hours to run on a standalone DBMS even for a small dataset like Miami.

Chapter 3

Modeling Epidemiological Case Studies in Distributed Database

In this section, we describe how interventions are expressed and implemented in the existing database-based framework. As stated earlier, interventions are the infection control and preventive measures to reduce the disease transmission in a contact network. Applying interventions, in a contact network, may change one or more properties of a set of individuals or even change the contact network. The interventions could be either pharmaceutical or non-pharmaceutical. Some of the pharmaceutical interventions include voluntarily taking vaccination, wearing a face-mask, antiviral, and so forth. While non-pharmaceutical include social distancing, cutting down non-essential activities, and so forth. In a graph dynamical system where an edge represents an interaction, and a vertex represents an individual, applying pharmaceutical interventions change the vertex properties(susceptibility of a node, infectiousness, etc.) while non-pharmaceutical interventions change the social contact network through edge addition or deletion. Hence, a range of intervention scenarios are possible, some of which have been realized and studied using the current Indemics framework. In the current system, the subset of vertices or edges to be changed is computed within a database.

Modeling Interventions using Tables: The data stored in the database can be classified into two main types: base and dynamic. Base tables contain data corresponding to the demographic attributes associated with an individual or social contact network that represents the interaction between people. Dynamic tables contain temporal data (also known as dendogram) about the disease transmission network i.e. information about when an individual gets infected, individual who transmitted the infection, and so forth.

Expressing interventions using SQL: A given intervention scenario is specified using SQL and typically consists of a set of 3-5 queries. These queries are executed one after another, each day, to find the subpopulation(subset of edges or vertices) to be intervened. An intervention consists of three steps [8]:

- **Compute** the individuals(vertices in the graph) S using a function I. I is a function of relational data R and the Dendogram up till time t, denoted by D_t.

- **Apply** an evaluation criteria in the form of function F over subpopulation set(S). For example, F could compute whether $|S| > 5\%$ of the total population.

- An **action** function H that computes the set $S = (S_1, A_1), (S_2, A_2),....$, where S_i is a set of individuals and A_i denotes the action that changes the attributes associated with individuals.

The functions F, I and H can be expressed using an SQL-like scripting language. Thus, the computational complexity of an intervention depends on complexity of evaluating the SQL queries for the three functions.

Over the past few years, different interventions have been studied using Indemics, some of which have been listed in the Table 3.1.

Table 3.1: Epidemiological case studies

Intervention Study	**Description**
Targeted	provide antiviral to diagnosed people
Triggered	apply intervention to protect some subpopulation universally
Household	all household members take preventive actions if anyone in the household is diagnosed
Sick leave (honest case)	the diagnosed employees in the companies complying the policy will stay at home from the diagnosis day to the recovering day
Sick leave (rational case)	the diagnosed employees in the companies complying the policy will stay at home for the max sick-leave period
Block	vaccinate all people in any block if the fraction of diagnosed people in that block exceeds some threshold
School	close any school if the fraction of diagnosed students exceeds 5%
Distance-1	close any school if the fraction of diagnosed students exceeds 5%

All the intervention studies listed in the above table, consist of a bunch of SQL queries that are executed every day to find the subpopulation to be intervened. Out of this list, we will discuss in details the following three scenarios: block vaccination, school closure and distance-1 vaccination. We describe their schema and the actual intervention queries in the three studies. These three have been chosen as they represent some of the most challenging scenarios in terms of intervention query operations and some of them being data-intensive in nature. These three studies are of varying complexities viz. block vaccination, school closure and distance-1 vaccination and present a mixed workload for efficient evaluation of our distributed database-based framework.

The queries are written in a Client script file in an SQL-like scripting language as described in [8]. The intervention script typically consists of SQL query operations which update the corresponding schemas.

3.1 Block Vaccination

In the block study, vaccination is provided to all residents of a census block group if an outbreak is observed in that particular block group. Here, the intervention is **triggered** when the fraction of diagnosed people in a block exceeds a set threshold, say, 5%. When the intervention is triggered, the form of **action** taken is vaccinating all the residents of the affected block.
Block Study Implementation:

1. Create a Block_Person table in the database for the block-person relation.
2. Specify the block intervention using the SQL-like language in a script.
3. Run the script along with the algorithm simulating contagion-diffusion.

Table 3.2 gives a list of the Base and dynamic tables used in block intervention.

Table 3.2: Block intervention schema

Schema	Description
BlkPop(block_id, person_id)	People residing in a block.
Diag_cases(person_id, day)	People diagnosed per day.
BlkDiagDailyCount(person_id,diagnosed_count, day)	Newly diagnosed people on the current day.
BlkInfWinAgg(block_id, diagnosed_count,day)	Count of sick residents in each block. individual stays in sick status for 6 days after diagnosis.
BlkIntvDaily(block_id, persons, intervened)	Infections in a block group on a particular day.

3.1.1 Block Intervention Script

In a given census block, if more than 5% of the people are diagnosed with a particular epidemic, then the intervention is applied to all people living in that block. This scenario is implemented as a series of five SQLs.

```
// Intervention step 1: Update the database with diagnosed people in a block on the current
    day.
Insert into Diag\_cases : people diagnosed on the current day
```

```
// Intervention step 2: Count the number of newly diagnosed people in a block on the current
     day.
Insert into BlkDiagDailyCount
(
        select count(b.person_id),block_id,"now" from BlkPop b, Diag_cases d
        where b.person_id = d.person_id and
                        d.day="now"
        Group By block
)
```

```
// Intervention step 3: Count the number of sick people in a block for the last six days.
Insert into BlkInfWinAgg
(
        select count(diagnosed_count), block_id, day from BlkDiagDailyCount
        where day > "now-6"
        Group By block_id
)
```

```
// Intervention step 4: Find blocks where over 5% people are sick to be intervened.
Update BlkIntvDaily(
        set intervened="now"
        where intervened = -1 and block_id in
        (
          select s.block_id from BlkInfWinAgg s, BlkIntvDaily i
          where day = "now" and
           s.block_id = i.block_id and
                 (1.0 * s.diagnosed_count/i.persons > 0.05)
        )
)
```

```
// Intervention step 5: Find people in the blocks identified in the previous step and apply
    vaccination to them.
Insert into intervention(
        select person_id from BlkPop
        where block_id in
        (
          select block_id from BlkIntvDaily,
          where BlkIntvDaily.intervened="now"
        )
)
```

3.2 School Closure

In the school study, a school is closed if an outbreak is observed in that particular school. Here, the school intervention is **triggered** when the fraction of diagnosed students exceeds a set threshold, say, 5%. When the school intervention is triggered, the form of **action** taken is closing the school.

School Study Implementation:

1. Create a Student-School table in the database for the student-school relation.

2. Specify the school intervention using the SQL-like language in a script.

3. Run the script along with the algorithm simulating contagion-diffusion.

Table 3.3 gives a list of the base and dynamic tables used in school intervention.

Table 3.3: School intervention schema.

Schema	Description
Student(student_id, school_id)	Students in a school.
SchDiagDailyCount(school_id, diagnosed_count, day)	Newly diagnosed students on the current day.
SchInfWinAgg(school_id, infections, day)	Count of sick students in a school. A student stays in the sick status for 3 days after diagnosis.
SchIntvDaily(school_id, students, intervened)	School with sick students exceeding a fixed threshold.

3.2.1 School Intervention Script

In a given school, if more than 3% of students are diagnosed with a particular epidemic, then the intervention is applied to that school. This scenario is implemented in the client as a series of five SQLs.

```
// Intervention step 1: Update the database with the diagnosed students in a school on the
    current day.
Insert into Diag_cases : students diagnosed on the current day
```

```
// Intervention step 2: Count the number of newly diagnosed students in a school on the
    current day.
Insert into SchDiagDailyCount
(
        select count(student_id), school_id, "now"
        from Student s, Diag_cases d
        where s.student_id=d.person_id and
                day="now"
        Group BY school_id

)
```

```
// Intervention step 3: Count the number of sick students in a school for the last three
    days.
Insert into SchInfWinAgg
(
        select count(diagnosed_count), school_id, day from SchDiagDailyCount
        where day > "now-3"
        Group By school_id
)
```

```
// Intervention step 4: Find schools where over 3% of students are sick to be intervened.
Update SchIntvDaily
(
        set intervened=-1
        where intervened = -1 and school_id in
        (
                select i.school_id from SchInfWinAgg s, SchIntvDaily i
                where day = "now" and
                        s.school_id = i.school_id and
                        (1.0 * s.infections/i.students> 0.03)
        )
)
```

```
// Intervention step 5: Apply intervention to the schools identified in the previous step.
Insert into intervention(
        select school_id from Student
        where SchIntvDaily.intervened="now"
)
```

3.3 Distance-1 Vaccination

In the distance-1 intervention study, an individual voluntarily takes vaccination if his neighborhood is infected. Here, the intervention is **triggered** when over 3% of the neighbors of an individual are diagnosed. When the intervention is triggered, the form of action taken is voluntary Vaccination by individuals with an infected neighborhood.
Distance-1 Study Implementation:

1. Create a Person_Person contact network table in the database that represents people-people interaction.
2. Specify distance-1 intervention using SQL-like language in a script.
3. Run the script along with the algorithm simulating contagion-diffusion.

Table 3.4 gives a list of Base and Dynamic tables used in distance-1 intervention.

3.3.1 Distance-1 Intervention Script

In a given social contact network, if more than 3% of the contacts of a person are diagnosed with a particular epidemic, then the intervention is applied to that person. This scenario is implemented in the client as a series of five SQLs.

```
// Step 1: Update the database with diagnosed people in a contact graph on the current day.
Copy to Diag\_cases : people diagnosed on the current day
```

Table 3.4: Distance-1 intervention schema

Schema	Description
D1Social_network(head, tail)	Social contacts between the heads (person id) and the tails (person id).
D1DiagDailyCount(person_id, diagnosed_count , day)	Newly diagnosed neighbors of an individual on the current day.
D1InfWinAgg(person_id, neighbors, day)	Count of sick neighbors. Individual stays in the sick status for 3 days after diagnosis.
D1IntvDaily(person_id, neighbors, intervened)	Individual with sick neighbors exceeding a fixed threshold.

```
// Step 2: Count the number of newly diagnosed neighbors of an individual on the current day
    .
Copy to D1DiagDailyCount
(
        select count(head), tail, "now"
        from D1Social_network n, Diag\_cases d
        where n.head=d.person_id and
                day="now"
        Group BY tail

)
```

```
// Step 3: Count the number of sick neighbors of an individual for the last three days.
Insert into D1InfWinAgg
(
        select count(diagnosed_count), person_id, day from D1DiagDailyCount
        where day > "now-3"
        Group By person_id
)
```

```
// Step 4: Find the individuals with over 3% sick neighbors to be intervened.
Update D1IntvDaily
(
        set intervened=-1
        where intervened = -1 and pid in
        (
                select n.pid from D1InfWinAgg s, D1IntvDaily n
                where day = "now" and
                        s.pid = n.pid and
                        (1.0 * s.diagnosed_count/n.persons > 0.03)
        )
)
```

```
// Step 5: Apply intervention to the individuals identified in the previous step.
Set Intervention:
        select pid from D1Social_network
        where D1IntvDaily.intervened=-1
```

Chapter 4

Literature Review

4.1 Distributed Databases in Scientific Computing

Distributed relational databases running on a large cluster of nodes was used to address data processing needs of astronomy [16]. In a series of study [19], authors show how relational databases can dramatically simplify the use of analysis over finite element meshes (FEM): a well-established tool used across many engineering disciplines. The FEM simulations can be very large and produce massive amounts of data across several machines. File-based data management is no longer feasible for such simulations. In both the disciplines, the access happens in bulk reads and write, and manipulation operations are much simpler and structured. Another interesting work by Givelberg et al. is an implementation which facilitates the use of parallel databases in scientific computing. They implemented MPI-DB [15], a software library which provides a parallel database service for large-scale scientific computations. Their work focused on leveraging the massive parallelism offered by large databases for scaling the computations. There also has been work featuring object-oriented databases in supporting scientific applications making use of complex objects and operations [25]. The motivation behind their work was a far more expressive, data modeling capabilities of OODBs for modeling such scientific structures like molecules, crystal units, etc.

4.2 Simulations and Databases

Modeling large and complex systems such as astronomy, epidemiology and biological reactions entails production and consumption of humongous amount of data. Accurate simulation based predictions demand sufficiently large data, although that is certainly not the only requirement for robust decision-making as explained in [13]. Yet the ever increasing scale and complexity of simulation parallels rapid growth in computational power and data. This increase is even more so in case of agent-based simulations for contagion simulation [3, 5, 9].

Agent-based simulations have gained importance in modeling disease dynamics with its roots dating back to 1970's [35]. Today, they still prove to be an effective way for simulating contagion diseases. Such simulations involve assessing the effects of autonomous interacting agents (individuals or groups) on the system. For systems involving a large number of agents, the processing power and data requirements are naturally more. For instance, in Episimdemics, an agent-based simulation framework [5], which models the disease spread over large social contact networks. Its input is a bipartite graph of person-location nodes and agents are the location nodes. Dealing with such data-intensive simulations necessitated efficient data management which served as the motivation behind incorporation of data-management technologies within a simulation environment. Several notable works in this area include Monte Carlo Database System [22] that allows analyst to attach arbitrary stochastic models to a database and computes approximate query results. Another work on probabilistic databases is ENFrame [39], a data processing platform based on probabilistic data mining and querying computes exact and approximate query results with error guarantees. SimSQL [10], an extension of MCDB, supports stochastic analysis by allowing querying of database-values Markov-chains. It has application in large stochastic agent-based simulations.

There has also been work in optimizing the simulation runs by optimizing the underlying simulation queries. One of the approaches to optimize the queries is result-caching technique [33] which is specific to stochastic composite simulations. In this work, the results of a stochastic composite model are cached, and all the subsequent query processing uses the result from the cache.

Another research line that separates the simulation and data-management by decoupling the compute-intensive and data-intensive tasks is Indemics [8]. Indemics has an HPC-based application for simulating the contagion-diffusion and a standalone data-engine which is queried to compute the interventions which are then supplied to the simulation engine. There are others [6, 17, 18, 38, 41] who have tried to solve the data management issues in simulations by using databases and have also studied the scalability of such systems. Another work by V.Suryanarayanan et al. 2013. is efficient realization of Range-Queries in a large-scale, distributed framework for parallel discrete-event simulations [36]. Our work is the extension of the Indemics framework by replacing the standalone database with a distributed database that supports massive query parallelism. To the best of our knowledge, no other work features the integration of a distributed data-engine and HPC-based simulation engine in the same supercomputing environment.

Chapter 5

Problem Statement

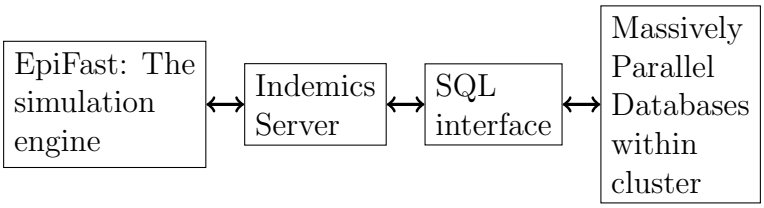
Simulating realistic intervention scenarios in epidemiology can be both data and compute intensive. Examples of such interventions are distance-1 and distance-2 studies. The amount of data and computation involved in such simulations can be overwhelming for a standalone system. There has been work in massively parallel execution and scaling of epidemiology simulations [29, 31, 44]. Some of these platforms do support interventions [29, 44] but none of these are database supplied interventions. To the best of our knowledge, Indemics is the only framework in which interventions are computed in the database and supplied to the simulation engine through an intermediate server.

In Indemics framework, the simulation engine runs in an HPC environment and hence the contagion-diffusion scales easily to few hundred cores. However, in most of the simulations with complex interventions such as honest and distance-1, a large fraction of total simulation time is spent in database access. The DBMS, that was supporting the Indemics framework earlier was a standalone database with limited resources. Running a simple intervention like Block on the Oracle database would take hours for medium-sized datasets such as Chicago and Boston. Often, multiple cells of a simulation are run for robust decision-making since the results are probability based. Although these cells can run in parallel in an HPC environment, the interventions still need to be computed inside a standalone database. This further degrade the simulation performance.

It is, therefore, essential in such type of simulations that we scale the interventions in order to improve the overall simulation time.

This work addresses the problem of scaling the data engine by using open source, distributed database on supercomputers. The distributed database partitions the data and supports parallel query processing which greatly improves the intervention performance. Additional advantage in using a distributed, open source database is that it enables us to make simulation specific modifications. It also allows optimization of the database that can improve the query performance.

In this thesis, we propose a new IndemicsXC framework, components of which are as follows:



In this framework, both EpiFast and data engine reside within the same supercomputer. The direct advantage is the use of the supercomputer resources to accelerate epidemiology simulations and increase throughput of a study. Henceforth, we will refer to this integrated framework of Indemics and distributed database as IndemicsXC.

Specifically, we discuss the following topics in the next chapters:

- In Chapter 6, we introduce a distributed, and, heterogeneous computing architecture for IndemicsXC. We present IndemicsXC with a distributed database structure that replaces the RDBMS component of the Indemics framework and has the capability to run large-scale simulations.
- In Chapter 7, we propose data distribution, execution strategy, and optimizations to the IndemicsXC systems for scaling multiple simulations running simultaneously. Such multiple run scenarios were not earlier feasible on the Indemics framework. We also demonstrate how this improves the performance of the current framework with respect to interventions like block, school and honest. We also discuss various performance parameters and their tuning, customized as per our system. Some of these parameters include query plan, amount of main memory for query processing, indexes, cache size on our supercomputer.
- In Chapter 8, we benchmark and compare the performance of IndemicsXC with Indemics for running single and multiple parallel simulations and study the throughput.
- In Chapter 9, we present a cost model for the intervention studies. We discuss different cost functions for predicting the query time in an intervention. The model predicts the query performance and the best configuration to run an intervention.

Chapter 6

PGXC Distributed System

6.1 The PGXC Distributed Database System

The DBMS used in our framework is a distributed database based on shared-nothing architecture with all the communication via the network interconnect. It is implemented as a master and slave style application. The master is also termed as coordinator and the slaves are termed as datanodes. Figure 6.1 is an illustration of the distributed DMBS framework.

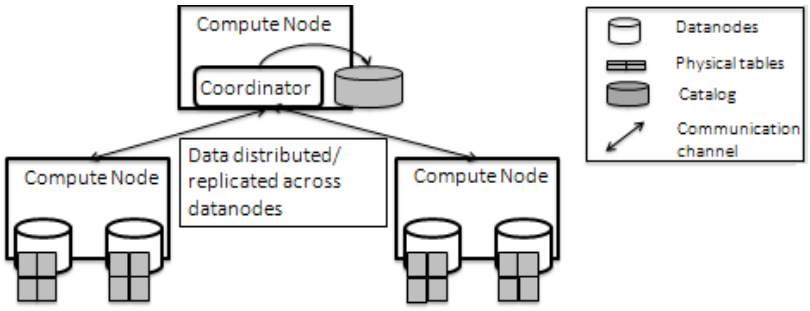


Figure 6.1: PostgresXC setup in an HPC environment. The coordinator or the master node is instantiated on one of the cluster nodes and the datanodes are started across multiple cluster nodes. User data is either replicated or distributed on each cluster node.

The PGXC system broadly consists of three components: the GTM or Global Transaction Manager, the Coordinator and the Datanodes. The GTM is timestamp server that provides transaction id's to all the transactions in the system. The coordinator is the frontend of the system. Clients connect to the coordinator to submit SQL queries. The datanodes are the storage hub. The physical data resides within the datanodes. The coordinator and the datanodes use the GTM service to implement distributed transactions. Our setup has one coordinator and multiple datanodes (distributed across the compute nodes).

6.2 Creating a PGXC Instance on Supercomputer

Before diving into the configuration of PGXC on the supercomputer nodes, we will discuss creating a PGXC instance on supercomputer. The PGXC has three components: the GTM, coordinator and the datanodes which have been described in the earlier section. The supercomputer has cluster nodes, each having memory and multicore processing units. Typically, GTM and coordinator are instantiated on a single compute node while the datanodes are spread across multiple physical nodes. The number of datanode processes to be spawned depends on factors like size of experimental dataset and the intervention complexity (data/ computation). There are two possible ways of deploying PGXC into the supercomputer cluster:

- **Multinode-Multicore setup**: This is a distributed setup in which both the coordinator and the GTM are deployed on a single compute node and the datanodes are scattered across multiple compute nodes. The supercomputer compute nodes used for this type of setup have relatively *less* number of CPU cores.
- **Singlenode-Multicore setup**: In this setup, all the PGXC components are co-located on a single compute node. The nodes in the supercomputer cluster with *high* CPU and memory resources are used for this kind of setup. The nodes in this setup are also called as the *fat* nodes.

6.3 Data Distribution

A database, in traditional setting, is a collection of tables along with its primary key and primary key-foreign key relationships. In a distributed database, an additional aspect that needs addressing is the manner in which the tables are distributed across the datanodes. We first discuss the distribution strategies available within PGXC and then we describe a mechanism for processing queries using the distributed/replicated tables.

A table could either be distributed or replicated across the datanodes. In the replication technique, each row of the table will be replicated to all the datanodes of the PGXC cluster. Distribution in PGXC means horizontally partitioning a table and spread the rows across some or all the datanodes. There are three distribution strategies provided by PGXC such as Hash, Modulo and RoundRobin. In Hash partitioning, a hash value calculated on a specific column is mapped to a datanode. Each row of the table is then distributed on the datanodes based on this computed hash value. In Modulo distribution, modulo of a specific column is computed, and the rows are distributed based on the modulo value. RoundRobin is the simplest distribution strategy in which the rows are distributed to the datanodes in a round-robin fashion. The mapping of a row to datanode is independent of the value of any specific column. If none of the strategies is mentioned while creating a distributed table, a column

with all unique values is chosen as the partitioning key. If there is no such column, then the table will be distributed on any random column.

6.4 Distributed/Parallel query processing: Inter-Query Parallelism

We now turn to the main aspect of PGXC, the ability to perform distributed processing of queries over a given set of tables. PostgresXC offers inter-query parallelism in which the same query(or subquery) is executed on all datanodes in parallel(See Figure 6.2). In the figure, the same query runs on all three datanodes. The type of data distribution (hash, modulo, or round robin vs. replication) for the tables and the type of predicates (join, groupby, filter) in the query determines the query processing strategy. Since a detailed discussion on distributed query processing is beyond the scope of this thesis, we confine our discussion to the type of queries used in intervention of various epidemiological studies (refer Chapter 3). These include queries with at most one of the following clauses: join, group by, update and insert.

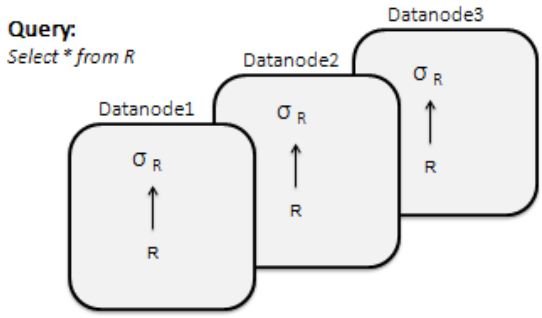


Figure 6.2: PGXC: Inter-query parallelism. The coordinator sends the same query fragment to all datanodes for execution. The datanodes carry out the query execution on the locally stored partition of the table.

Processing of such queries in the PGXC system typically involves the processing engines present at the both datanodes and the coordinator. The coordinator builds the distributed query plan and orchestrates the query processing. It finds the datanodes required for answering the query based upon the distribution of the tables involved in the query, and the semantics of the query. It finds a fragment of the query that is then sent to the datanodes. The datanodes, in parallel, process the queries and ship the results to the coordinator. After receiving the results, it performs post-processing i.e., execute a query fragment on the results from the datanode to obtain the final result. Complex queries may involve several rounds of processing back and forth between the datanode and the coordinator. We say the fragment of the query being executed at the datanodes as being *shipped*.

The following example and supporting figure 6.3 illustrate the processing of a given query consisting of a simple join followed by aggregate. Both the tables in the example are hash distributed on the *person_id* column.

```
// Sample query
select count(b.person_id),block_id,"now" from Block_Pop b, Diag_count d
        where b.person_id = d.person_id and
                        d.day="now"
        Group By block
```

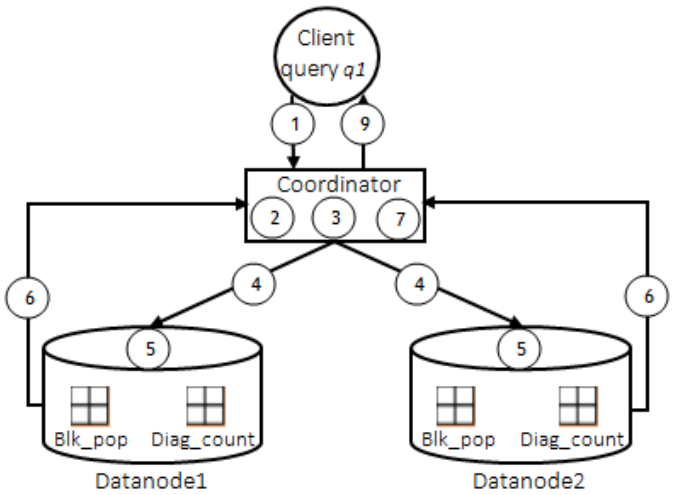


Figure 6.3: Typical query processing steps in PostgresXC, ○ - denotes the query step and ⊞ - denotes fragment of a distributed table.

Step No.	Operation
1	Query q1 sent to the PGXC coordinator
2	Query analysis at coordinator
3	Coordinator locates the datanodes having tables Block_pop and Diag_count
4	Coordinator ships the remote query or query fragment q1 to datanodes 1 and 2. In this case, query fragment is the complete query i.e. group by happens at both coordinator and the datanodes.
5	Datanodes execute q1 locally
6	Datanodes send local result to the coordinator
7	Coordinator does an aggregate of the output from all datanodes
8	Coordinator returns the results to the client

In the figure 6.3, query processing steps are numbered from 1 through 8 in order. The process starts at the coordinator where the query is analyzed and planned (steps 1 and 2).

The coordinator fetches the metadata to locate the datanodes having the rows of the tables mentioned in the query (step 3). The coordinator then checks if the query can be processed entirely on datanodes and if so, sends the entire query to the datanodes (step 4). If not, then the coordinator sends a query fragment to the datanodes (this scenario is not illustrated in the above example). On receiving the query, the datanodes locally execute it and send the results back to the coordinator (steps 5 and 6). The coordinator performs post-processing on the results if necessary and sends the aggregated results to the client (steps 7 and 8).

6.5 Query Shippability

A short note on shippable query (offloading the query (or subquery) to datanodes):
A query (or subquery) is shippable when all the data required to evaluate it is present on the datanodes. We illustrate a few template queries (or subqueries) when this condition holds true for two sample tables A(a) and B(b) which are hash distributed over respective columns a and b respectively.

Shippability criteria 1: A simple select with/without projection on either relation A or B.

Shippability criteria 2: A join of A and B with equality predicate $a == b$.

Shippability criteria 3: A join of A and B with equality predicate $a == b$ and aggregate on column a.

Shippability criteria 4: An update operation conditioned on a subquery (given that the subquery satisfies either of the shippability criterions).

Shippability criteria 5: An insert operation conditioned on a subquery (given that the subquery satisfies either of the shippability criterions and insert pushdown is feasible).

In addition to the above criterion, a shippable query also has to meet criterion such as lack of global constraints, local triggers, local materialization, etc. A query that meets the above criterion may be completely offloaded to the datanodes for execution. For such a query, the optimizer/planner pushes the query operations to the datanodes where it executes in parallel.

6.6 Tuning PGXC instance on supercomputer for simulations

Distributed databases come with many configuration parameters. Their values are used to tune the reliability and limits (on resource utilization) of the database system. Both these aspects have a profound impact on performance. The limits dictate the kinds of plans (sort merge, hash, or, bitmap) feasible for joins and aggregates queries. In general reliability has an inverse correlation with performance, i.e., increased reliability comes at a cost of reduced performance.

Two criteria guide the choice for the parameter values. First, to trade-off reliability for increased performance and second, to most effectively utilize supercomputing hardware resources. This configuration for the database is atypical and most likely novel. That is because databases are mostly used in enterprise, web, or warehouse domains and running on a cluster where on one hand reliability is critical while at the same time, the other chances of failure are high. In our scenario reliability is not important because in case of failure we could always restart the simulation. Secondly, supercomputers are far more reliable than typical small clusters.

Listed below are the settings we modified for the better performance of our framework:

- **Using Tmpfs** - Memory Based File system: tmpfs is the temporary file storage facility on Unix like operating systems and uses the operating system page cache for storing file data [32]. All the dynamic tables created in the simulation are stored on tmpfs. For instance, for Miami- Block scenario, using tmpfs instead of local disk for storing the dynamic tables gave us a performance gain of 2.5X.
- **Increasing work memory**: Work memory specifies the amount of memory to be used by internal sorts and hashes before writing to temporary disk files. The intervention queries in our case are complex with several sorts and hash operations and for larger datasets, memory usage for these operations tend to go up.
- **Using Copy operation for bulk inserts**: Copy utility moves data between standard file-system and database tables. In our case, using COPY in place of bulk-inserts doubled the system performance.
- **Increasing Checkpoint segments**: Checkpoint segments determine the maximum log file segments between Write Ahead Log checkpoints [1]. This parameter affects crash recovery and must be set high in cases of heavy transactions. The default is 3 segments (each segment is about 16MB). In our case, we set this value to 128 which gave us 1.2X performance improvement.
- **Increasing Shared Buffers**: Shared buffers set the amount of memory the database server uses for shared memory buffers. The default value of this parameter is 32 MB. We set this value according to the data-size mentioned in the results section. Setting this parameter to according to the dataset size results in considerable performance gain.

- **Turn off Sequential Scan**: Setting sequential scan to off forces the query planner to use Index scan as opposed to sequential scan.
- **Using clustered indexes**: We made use of clustered indexes for clustering large demographic tables according to the indexes we created for them. Clustering based on indexes is a one-time operation and so any updates to the table are not clustered. Hence, we only clustered read-only tables.
- **Vacuum Analyze**: Doing Vacuum analyze after updates/inserts is necessary as such operations alters the data distribution. It ensures that the query planner has the updated statistics for each table that it uses at the time of planning. In addition, vacuum analyze reclaims the space by cleaning up the dead tuples which can be reused.

In addition to the above optimizations, we also made some simulation specific changes to the PGXC code pertaining to the queries that modify the relational table data. More precisely, the queries involving inserts and updates. These changes will be elaborated in the next section.

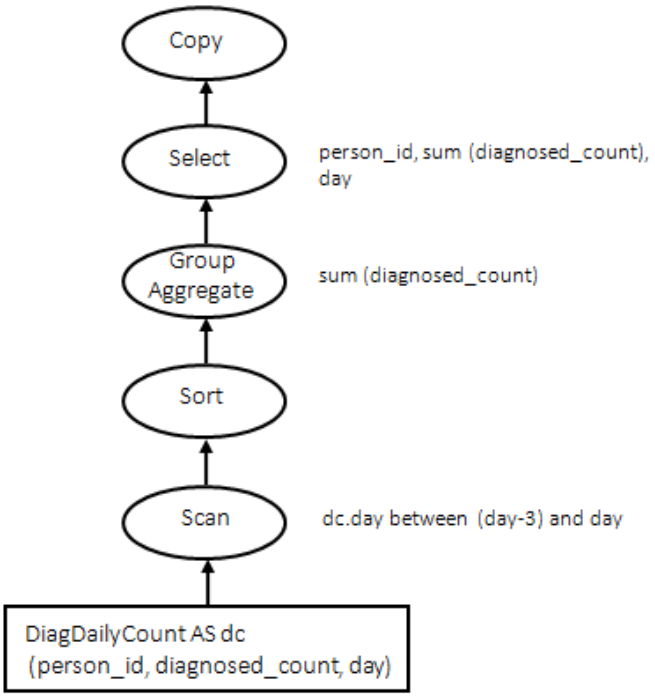
6.6.1 Short note on impact of work memory on Aggregates

To illustrate how work memory can impact the optimizer, we present the planner output of one of the queries from the Distance-1 intervention for Miami dataset. Below we show the query and the plans chosen for two values of work memory 64MB and 512MB.

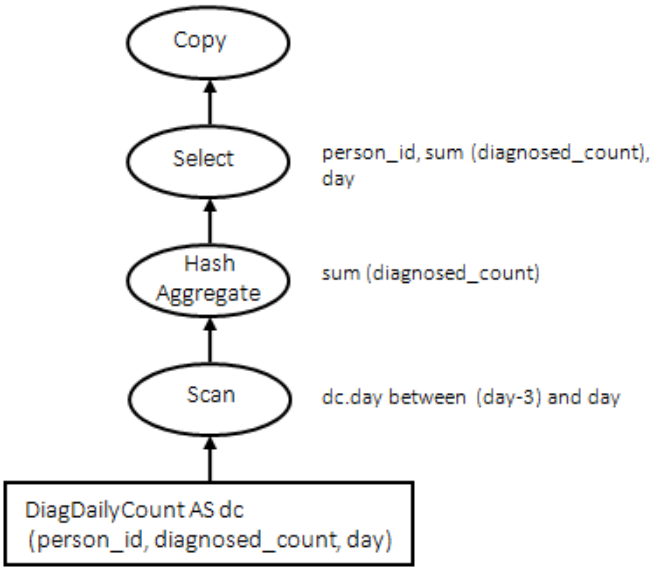
```
// Sample query
select person_id, sum(diagnosed_count),"day" from DiagDailyCount dc
        where day between (day-3) and day
        Group By person_id
```

From figure 6.4, we see that the optimizer chooses different query plans for the given values of work memory. The plan in both the cases start with scanning of the relation DiagDailyCount but differs in the type of the aggregate function chosen. For a small value (64 MB) of work memory, the optimizer decides to go with GroupAggregate algorithm that requires presorted data. Once sorted, it aggregates the data in one scan over the result set. Since the sorting algorithm (merge sort) does not store the entire result set in memory, this plan requires relatively small amount of work memory. The time complexity for this sort-based aggregate for n rows would be $\mathrm{O}\big(n \log(n)\big)$. Sure enough, for the above query, hash-based approach performs 1.38X better than the sort-based approach. In contrast, for high value of work memory (512MB), the optimizer chooses HashAggregate, that tends to buffer the entire result in memory. HashAggregate does not require sorting, and it simply hashes each row in a hash table. Hence, the total time complexity for hash-based aggregate function is linear, i.e., $\mathrm{O}\big(n\big)$.

The preferred plan in our case is HashAggregate since our systems have enough memory (per supercomputer node) for doing the hash-based joins and aggregates. Thus, in summary,



(a)



(b)

Figure 6.4: Query plans for different values of work memory parameter. (a) Canonical query plan for 64MB Work memory: Sort-based aggregate. (b) Canonical query plan for 512MB Work memory: Hash-based aggregate.

the performance parameters provided by PostgresXC need to be tuned to extract the best performance out of it. The parameters need to be modified as per the system configuration to get the best results.

Chapter 7

Optimization Techniques in the Distributed Database

7.1 Efficient data distribution strategy for parallel query processing in intervention driven simulations

As discussed, data distribution strategy affects the query plan which is the primary factor for scalability and performance. In general, plans that push down the query to datanodes scale better. Here we discuss the distribution for static and dynamic tables involved in the simulation for block, school and honest studies. Our objective is to push down the largest possible fragment of the query to datanodes. Table 7.1 shows the distribution (along with the distribution column) vs. replication strategies for block, school and distance-1 intervention queries.

Given the distribution strategy, we now take block study as an example and describe the distribution plan of each of the SQL in the intervention script (refer Section 3.1). In particular, we look if the query is being pushed down completely or partially. As mentioned before, block intervention queries involve different kinds of query operations. We consider a query being pushed down if all its operations (join, aggregate, select, etc.) get pushed down to the datanodes. For block study, we replicate the static *BlkPop* table (refer Chapter 3.1) and distribute all the dynamic tables as given in the above table. Below, we have categorized the block intervention queries based on the operators (such as select, join, groupby, insert and update) present in them.

- **Insert queries (Intervention steps** 1**,** 2**,** 3 **and** 5**)**: In the block study, intervention steps 1, 2, 3 and 5 contain the insert operator. In all the modern DBMSs, an insert operation is inherently sequential. Same is the case with the distributed database where inserts happen sequentially. The coordinator ships each incoming row to the

Table 7.1: Different data distribution strategies for Block, School and Distance-1 intervention scenarios. The tables are either distributed or replicated based on their sizes and also the queries in which they occur. The table distribution strategies affect the operation push down in the intervention queries and hence is important to study.

Study type	**List of distributed tables (distribution column)**	**List of replicated tables**
Block	Diag_cases (person_id) BlkInfWinAgg (block_id) BlkDiagDailyCount (block_id) BlkIntvDaily (block_id)	BlkPop
School	Diag_cases (person_id) SchDiagDailyCount (student_id) SchInfWinAgg (school_id) SchIntvDaily (school_id)	Student
Distance-1	Diag_cases (person_id) D1Social_network (head) D1DiagDailyCount (person_id) D1InfWinAgg (person_id) D1IntvDaily (person_id)	

respective datanodes, which in turn add it to the table partition. That poses a major bottleneck for scaling insert queries as the insert operation does not get pushed down. We modified the insert operation in the database such that inserts, when feasible, are completely shippable to all datanodes. It allows the query to be pushed down and also has an added advantage that it can run in parallel on all datanodes. Thus, according to the shippability criteria 5, intervention steps 1, 2, 3 and 5 of the block study involving insert operation get pushed down to the datanodes.

- **Select, Join, GroupBy queries (Intervention steps 2 and 3):** Intervention steps 2 and 3 from the block study involve select, join and aggregate operations. In the case of query 2, we distribute the join tables by the join predicate - *person_id*. But this query also involves a groupby on the *block* column. Hence, according to the shippability criteria 2, the join operation and *local* groupby gets pushed down to the datanodes while the overall aggregation (groupby) happens at the coordinator. Therefore, intervention step2 of block study gets partially pushed down to the datanodes. In query3, both the join and groupby happens on the *block* column, and therefore according to the shippability criteria 3, it gets entirely pushed down to the datanodes completely.

- **Update query (Intervention step 4):** Intervention step 4 is an update query with an inner select subquery. An update operation is also sequential that affects the query scalability. We, therefore, modified the update code in the database so as to make

the update operation completely shippable to the datanodes. This change, therefore, makes the update operation scalable as it gets pushed down and also helps in achieving parallelism. However, in a case where the update operation is at the top-level of the execution plan, and the inner sub-query(queries) cannot be evaluated completely on the datanodes, and then the update operation as a whole becomes non-shippable. In this case, however, the inner subquery involves a join on the column *block_id* which, according to the shippability criteria 2, gets pushed down completely. Hence, according to the shippability criteria 4, whole intervention step 4 gets pushed down to the datanodes completely.

- **Select, Join query (Intervention step 5):** Intervention step 5 involves a select and join between BlkIntvDaily and BlkPop on the *block_id* column. Since, we distribute BlkIntvDaily on block_id and BlkPop has been replicated on all datanodes, as per the shippability criteria 2, the entire query gets pushed down to the datanodes.

7.2 Task/Data distribution strategies

One of the objectives in our work is to increase throughput of a set of simulations (replicates of cells) within an epidemiology study by effectively utilizing the cluster/supercomputing resources. Given the choices for data distribution strategy within the PGXC system two natural strategies emerge for increasing throughput. In the both the strategies the intervention processing for all the simulations proceeds concurrently.

- **One-to-One or "Inter-cell parallelism"**: Intervention processing for simulation is independent of each other i.e., they are in separate processes and do not use shared memory.
- **All-to-All or "Inter and Intra-cell parallelism":** Intervention processing is handled by separate processes, but read-only (static table) data is shared.

The two distribution strategies are shown in the figures 7.1 and 7.2 below. The performance results for both the strategies will be demonstrated in the results section.

One-to-One strategy: In this approach, the read-only base tables are replicated across all datanodes. All the dynamic tables belonging to one simulation instance will be stored on a single datanode. Hence, we are mapping each cell or instance to one datanode as shown in figure 7.1. The plus side of using this approach is that there will be no bottleneck on the coordinator side as all the query operations belonging to a single instance will be pushed down to its respective datanode. Hence, there will be no need of aggregating the results from every datanode. The downside is that the queries won't be distributed across the datanodes and so we won't get any parallelism.

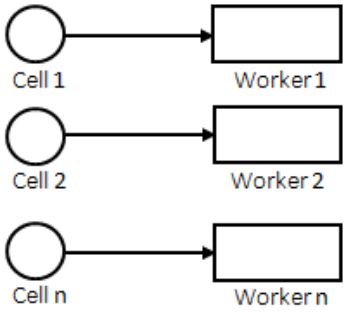


Figure 7.1: One-to-One: A strategy for distributing the data and query task on the PostgresXC system. In the one-to-one technique, the simulation data of a simulation cell is stored on one datanode rather than distributing it. Therefore, as per this setup, one datanode handles one simulation cell.

All-to-All strategy : In this approach, the read-only base tables are distributed across all datanodes. All the dynamic tables belonging to one simulation instance will be distributed across all datanodes. Hence, we are mapping each instance to all the datanodes as shown in figure 7.2. The plus side of using this approach is that we will be leveraging the benefits of parallelism as the query gets distributed on all datanodes. The downside is that there could be a possible bottleneck on the coordinator side as it has to aggregate the results from all datanodes. It could pose as an issue with the increasing number of instances and datanodes.

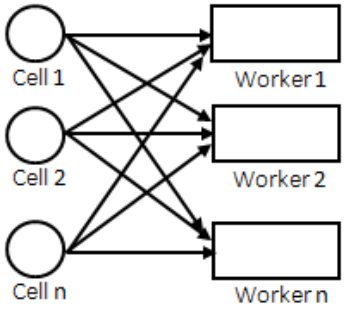


Figure 7.2: All-to-All: A strategy for distributing the data and query task on the PostgresXC system. In the all-to-all technique, the simulation data of a simulation cell is distributed across multiple datanodes. Therefore, as per this setup, multiple datanodes handle a simulation cell in parallel.

7.3 Partitioning Dynamic Simulation Tables

Almost all the modern databases (centralized and distributed) support table partitioning feature now-a-days [11, 34, 37]. Table partitioning refers to physical splitting of a table into

multiple smaller parts, though logically it is still a single table. The table can be partitioned either horizontally or vertically, however horizontal partition is the most common way of splitting a table. Partitioning a table also divides its indexes into smaller partitions so that it becomes easy to maintain a partitioned table.

The main motivation behind using partitioned table is the growing size of tables which can create a lot of problems such as difficulty in performing administrative tasks (updating indexes, cleaning up dead rows, statistics, etc.), increased data contention during query processing and many more. Hence, as the tables grows in size, there is also a linear increase in its access time. The use and benefits of using partitioned tables in open source systems like PostgreSQL and MySQL to optimize the database performance has been studied [2, 45]. There has also been work in developing optimization techniques for join queries over partitioned tables [20].

PGXC system, in addition to supporting physical (and logical) distribution of a table across multiple datanodes, also supports splitting of a table partition, residing on a datanode, into multiple physical parts. These parts would still be treated as a single logical partition of the table. Hence, PGXC supports partitioning a table both across the datanodes and on a datanode. In PGXC, a table can be either range or list partitioned. Range partitioning refers to splitting a table into ranges determined by the key column or set of columns. In list partitioning, one explicitly lists the key values in each partition. This can be illustrated as: consider the BlkInf table in Block study that is hash distributed on person_id column across 2 datanodes. Let's say we want to split the fraction of BlkInf stored on the two datanodes into three physical tables based on value ranges of person_id column. Assume that all values of the person_id column are between 1 and 100. When, we hash partition the table, the person_id column values on either of the datanodes could be between 1 and 100. If the person_id column values residing on datanode1, are in the range 1 to 100, the first partition could contain column values from 1 to 16, the second partition could contain values from 17 to 33 and so forth. Figure 7.3 shows the horizontal *hash* partitioning of BlkInf table (referred as B in the figure) across the two datanodes and the subsequent range partitioning on both the datanodes.

Partitioning tables in distributed databases and the associated optimization techniques has been studied which are mostly based on data-localization [28]. Data-locality is an important property of distributed databases that majorly impacts the query performance and scalability. During the query re-write phase, a "partition-aware" system would know on which node the data resides and this would help in the filtering rules. Based on the data distribution, the filtering rule will decide the partitions to be pruned. It also helps in pruning joins from the partitions that are unlikely to contribute to the query result.

Partitioning the simulation tables:
As the simulation proceeds, every day infections increase with a Gaussian probability. The ramification of increasing infections is the growing size of the simulation tables. For instance,

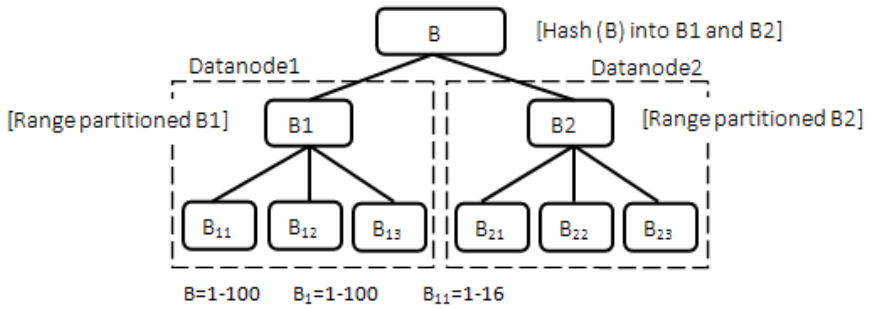


Figure 7.3: Horizontal distribution and partitioning of table B. The table is first distributed on a column'a' across the two datanodes and then again partitioned on a second column 'b' on the datanodes. Please note the difference between distribution and partition here. A table is distributed 'across' datanodes and a table segment on a datanode is further 'partitioned'. This method can be useful in a case where we are required to replicate a slightly big table. By partitioning the table on the datanodes, we can improve the query performance.

consider $D1IntvDaily$ table from distance-1 study which keeps track of a person with infected neighbors along with the number of infected numbers around him. As the infections increase, we observe that the size of the table also increases which degrades the performance of queries involving the table $D1IntvDaily$. Since there are indexes created on it, the performance hit is even more pronounced as the maintenance tasks (updating indexes, updating statistics, etc.) become an overhead. Although, we distribute the table by hash partitioning, one cannot assume skew-free distribution as the hash function does not distribute the rows evenly across the datanodes unlike roundrobin or range-based partitioning. The skew-effect can be even more pronounced in the case when distribution column values are not unique. For instance, the $D1Social\_network$ table contains person-person interaction data and hence there can be multiple duplicate values in either 'head' or 'tail' column. Since, PGXC does not provide range-based data distribution and as roundrobin distribution does not support "query pushdown", we decided to use hash distribution technique.

We address this problem by range partitioning the distributed table rows on each of the datanodes. Since we know that most of the queries access the infections for the current day, we can split the table based on the day range values. For illustration, consider the below query from distance-1 intervention:

```
// Sample query
Insert into D1InfWinAgg
(
        select count(diagnosed_count), person_id, day from D1DiagDailyCount
        where day > "now-3"
        Group By person_id
)
```

The above query selects the count of infections for the last three days from the $D1DiagDailyCount$ table. We will see its distribution and subsequent range-based partitioning.

As described earlier in section 7.1, $D1DiagDailyCount$ is hash distributed on column per-

son_id. After creating the "master" table, let's say we split it into 4 small tables. The four small partitions will be created and provided with table constraints as :

```
CREATE TABLE D1DiagDailyCount_1 (
    CHECK ( day >= 1 AND day < 30 )
) INHERITS (D1DiagDailyCount);
```

```
CREATE TABLE D1DiagDailyCount_2 (
    CHECK ( day >= 31 AND day < 60 )
) INHERITS (D1DiagDailyCount);
```

```
CREATE TABLE D1DiagDailyCount_3 (
    CHECK ( day >= 61 AND day < 90 )
) INHERITS (D1DiagDailyCount);
```

```
CREATE TABLE D1DiagDailyCount_4 (
    CHECK ( day >= 91 AND day < 120 )
) INHERITS (D1DiagDailyCount);
```

Hence, the above query will only be using few of the table partitions which will make the administrative tasks faster and will result in efficient pruning of unwanted partitions of the table and thus improve the query performance.

Chapter 8

Performance Evaluation

The strength of our distributed database solution is that it retains the simplicity of Indemics by using the same SQL query interface for expressing interventions. Therefore, users may find it easy to adapt to this framework without making any modifications to the simulation engine or the Indemics server. In this section, we compare the performance of the two systems - Indemics and IndemicsXC for implementing epidemic interventions. We designed and conducted a series of experiments to demonstrate the performance of IndemicsXC in simulating complex and realistic intervention scenarios. Specifically, we present experimental results for following two types of intervention scenarios i) fairly simple and supported by both IndemicsXC and Indemics (school, block, targeted, household) and ii) complex, data-intensive and only supported by IndemicsXC (distance-1). We evaluated the performance of IndemicsXC for both the scenarios. For the first type, we compare the performance of IndemicsXC to Indemics and for the second type, we present results that characterize the performance of IndemicsXC. For a fair comparison, we used same datasets and disease model parameters for both the systems such that they produce exactly the same outcome.

Plan of experiment study:
we conducted a set of experiments to support the claim that the distributed databases are a better option to implement the interventions than a standalone database. Our experiments analyze the performance of the IndemicsXC framework for a variety of intervention scenarios with different contact networks. We broadly classify the experiments into two categories: i) Single-cell and ii) Multi-cell. In single-cell experiments, we evaluate the performance of a single simulation run. While in multi-cell experiments, we measure the throughput by running multiple simulation cells in parallel.

In Section 8.1, we present a comparison between Indemics and IndemicsXC in terms of performance and throughput. That is, we present both single and multi cell results for the two systems. In Section 8.2, we have carried out performance analysis of IndemicsXC. We show how it scales to support complex intervention scenarios over large contact networks with millions of agents. Here, the present results for a single simulation run. In Section 8.3, we

present a throughput comparison for the two execution strategies(all-to-all and one-to-one) discussed in Chapter 7.2. In the throughput experiments, the variables are the number of cells, datanodes and the compute nodes. Here, we present results of multiple cells running in parallel. Section 8.4 presents the results depicting the effects of the optimization techniques discussed in Chapter 7 on the intervention query performance. In this section, we present results for a single simulation run.

Experimental Setup:
In our simulation framework, complexity of intervention queries depends on the scenario to be simulated, the contact network as well as the disease model parameters. Figure 8.1 gives the input parameters used in our experiments. We measure the performance of both the systems in terms of parameters given in Figure 8.2.

Table 8.1: Statistics about the datasets used in the intervention studies.

Region	**Population—V— (In millions)**	**Contacts—E— (In millions)**	**Block size**
Miami	2.1	53	1166
Chicago	9	262	6365
LA	16.2	460	10232

The experiments were conducted for synthetic population from three geographical regions: Miami, Chicago and Los Angeles(LA). Table 8.1 gives the statistics (—V— vertices and —E— edges) for these three datasets. We chose these three datasets as region has different variations in terms of population size, age distribution, household income and size distributions. The experiments have been designed and performed for simulating disease propagation in selected U.S. metropolitan areas with different intervention strategies viz. block, school and distance-1 to show the performance of IndemicsXC in realistic applications.

Table 8.2: Intervention query features in terms of static table size and type of operations involved in the queries.

Type of Intervention	**Table size (in terms of Population N)**	**Type of Queries**
Block	2*N	2-table-join, group-by, insert, update
School	2*N	2-table-join, group-by, insert, update
D1	50 * N	3-table-join, 2-table-join, group-by, insert, update

In the Chapter 3, we introduced the block, school and distance-1 intervention strategies which we studied in this thesis. Table 8.2 lists the features of these intervention strategies in terms of table size and the type of SQL for each intervention step.

1. **Random Seed:** Different infection seeds to make the output different.
2. **Number of iterations:** 5. We fixed the number of iterations to 5 as we observed small variance among the replicates for all the intervention studies. For instance, the average deviation for school and block studies are 7ms and 15ms respectively.
3. **Simulation duration:** All our experiments have been simulated for a duration of 300 days, unless otherwise stated.
4. **Intervention Types:** Vaccination, school closure.
5. **Trigger in the interventions:** Threshold based. For instance, close a school if more than 5% of students are diagnosed.
6. **Regions simulated:** Miami, Chicago and LA.
7. **Type of experiments:** Single and multiple cells.
8. **Data/work distribution strategies:** One-to-One and All-to-All.
9. **Simulation engine:** EpiFast.
10. **Simulation Environment:** High performance shared-nothing linux cluster. The cluster consists of 96 *small* and 2 *fat* nodes. Each small cluster node has Intel Core Xeon $X5670$ processors running at 2.93GHz, each with 6 cores. Small cluster nodes have 48GB of system memory (8 GB/core) per node. Each fat node has Intel(R) Xeon(R) processors running at 2.27GHz, with 10 cores. Each fatnode has 1TB of system memory (100GB/core).
11. **Distributed database:** Postgres-XC 1.2.1 cluster database running on the same linux cluster as EpiFast.
12. **Standalone database:** Oracle Database 11g running on a server with 16 CPU cores and 64GB RAM.
13. **System setup:** Multinode-Multicore setup (refer Section 6.2), unless Singlenode-Multicore setup is specified.

Figure 8.1: Input Parameters in the Experiments

1. **Run time** of the intervention queries.
2. **Scalability** - both weak and strong scaling. In weak scaling, datanodes are added to the database as the dataset size increases, whereas in strong scaling dataset size remains constant while the datanodes are added.
3. **Throughput** by running multiple simulation instances in parallel.

Figure 8.2: Output Parameters in the Experiments

We conducted both single and multi-cell experiments to evaluate the performance of IndemicsXC. Multi-cells experiments have been conducted by concurrently running multiple cells (see section 1.1.1 for reference) of simulation. The distribution of the tables involved in the simulation is as per the description given in section 7.1.

8.1 Performance and Throughput Comparison with Indemics

In this section, we provide a performance comparison of standalone and distributed database for intervention query processing.

8.1.1 Performance for query processing

For performance analysis, we carried out seven case studies, discussed in Chapter 3. Not all of them are feasible on Indemics such as Honest and Distance-1. We compared the query execution time on IndemicsXC with that on Indemics and compiled the speedup in Table 8.3. The runs were simulated on three regions- Miami, Chicago and LA. We have used the same disease model for all the comparison runs with similar infectivities. The results show that the distributed database-based framework indeed performs better than the current Indemics framework. we see a consistent speedup in all the case studies. The biggest improvements can be seen for the distance-1 intervention with a speedup of 20X for the Miami region. Though the performance improvement varies from region to region, the overall trend indicate increased intervention query processing performance of our framework.

8.1.2 Throughput comparison

Figure 8.3 and 8.4 presents a performance comparison between IndemicsXC and Indemics for simulating Block intervention over Miami and Chicago regions respectively. All parameters for the experiment are same as the default configuration except that we conducted this

Table 8.3: Overall execution time comparison between simulation runs over Oracle and PostgresXC. The results show that our setup performs better for simple and complex intervention queries and also for datasets of varying sizes and densities. This performance gain can be attributed to different performance parameters and optimizations discussed in Chapter 7

Study	**Region**	**Query time in Indemics (in mins)**	**Query time in IndemicsXC (in mins)**	**Performance Gain**
Targeted	Miami	0.6	0.07	8X
	Chicago	2.2	0.289	7X
	LA	4.11	1.08	4X
Household	Miami	6.16	3.11	1.9X
	Chicago	6.45	3.89	1.6X
	LA	16.91	10.7	1.5X
Sick leave (Honest)	Miami	65	16.18	4X
	Chicago	x	26.95	
	LA	x	31.19	
Sick leave (Rational)	Miami	1.12	0.163	6X
	Chicago	2.48	0.46	5X
	LA	4.89	1.08	4.5X
Block	Miami	6	2.92	2X
	Chicago	45	7.01	6.5X
	LA	107	12	3.5X
School	Miami	33.70	2.2	15X
	Chicago	90	4.1	22X
	LA	x	10.8	
D1	Miami	175.43	8.81	20X
	Chicago	x	22.12	
	LA	x	37	

Small 'x' in the simulation-time column signifies that Oracle cannot handle the given query.

experiment on the Singlenode-Multicore setup and for varying number of simulation cells. We fixed the number of datanodeas 2 for Miami and 4 for Chicago. The figure indicates that IndemicsXC gives better performance than Indemics as we go on increasing the number of simulation cells. The execution time trend for Indemics run shows a steep increase as the number of cells increase, while the increase in the execution time for IndemicsXC is much more gradual. This steep degradation of performance in the case of Oracle may be because of increased I/O since it is a standalone system with limited resources. These results suggest that the distributed database scales well for increasing number of simulation instances.

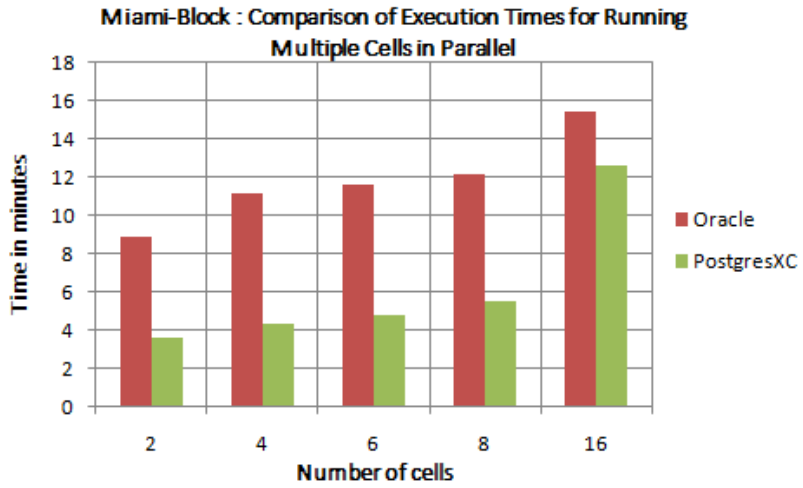


Figure 8.3: Performance comparison between Oracle and PostgresXC for simulating interventions in multiple cell scenario. The bar plot shows the total running time of simulation with varying number of parallel cells in Block intervention scenario. The simulated region is Miami.

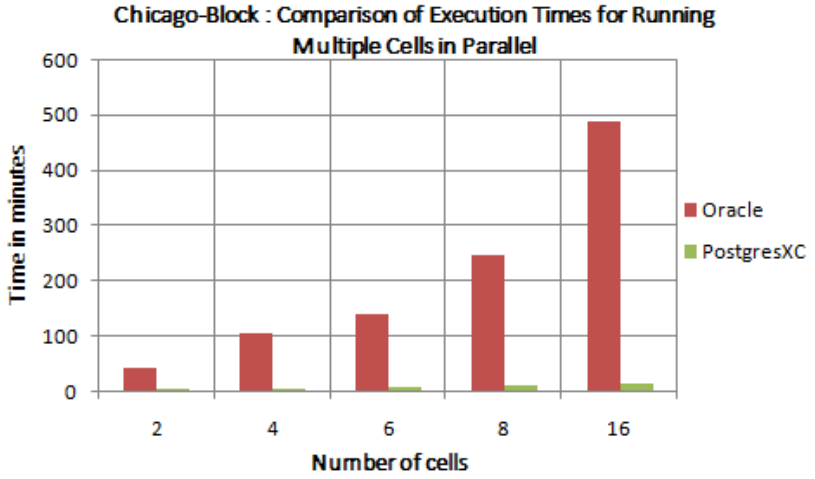


Figure 8.4: Performance comparison between Oracle and PostgresXC for simulating interventions in multiple cell scenario. The bar plot shows the total running time of simulation with varying number of parallel cells in Block intervention scenario. The simulated region is Chicago.

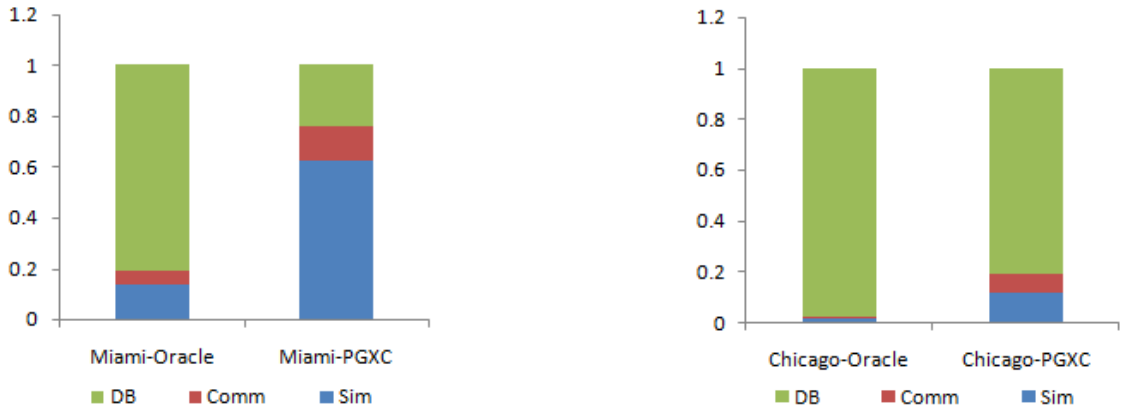
8.1.3 Simulation time breakdown

Given below is a breakdown of total simulation time of IndemicsXC into three components:

1. **SimTime:** EpiFast running time i.e., the cost of disease propagation simulation for each simulation day

2. **DBTime:** Database query time i.e., cost of intervention simulation for each simulation day

3. **CommTime:** Communication time i.e., the data communication cost between EpiFast and database.



(a) (b)

Figure 8.5: Total Simulation time breakdown of Indemics and IndemicsXC for block intervention. (a) Simulation time breakdown for Miami. (b) Simulation time breakdown for Chicago. The database query time in the block run over Chicago is dominant because of the increased query complexity with the contact network size.

In this experiment, all parameters are same as the default configuration with the exception that the number of datanodes has been fixed to 2 for Miami and 4 for Chicago. It is a single cell experiment on a single compute node. Figure 8.5 shows a time breakdown comparison of IndemicsXC with Indemics. It can be seen from the figure that IndemicsXC brought down the database time significantly for Miami. Also, in case of Chicago, the fraction of total time-spent in database access is less in comparison to the standalone oracle database. However, the database access time for Chicago is much more than the contagion-diffusion time. This difference could be attributed to large and dense contact network of the Chicago region which increases the query complexity. These plots emphasize the importance of optimizing the database access time to bring down the overall simulation run time.

8.2 Scalability of IndemicsXC in Complex Interventions

In our scalability experiments, we quantify the performance of the distributed database in the IndemicsXC system in terms of strong and weak scaling with increasing datanodes. For a given intervention scenario, strong scaling is achieved by fixing a dataset size and adding

datanodes to bring down the query execution time. On the other hand, weak scaling is achieved by adding datanodes as the dataset size increase.

8.2.1 Strong Scaling: Performance of Distance-1 intervention study

We show the performance results of simulating distance-1 study scenario with IndemicsXC. Before presenting the results, a brief note on the complexity of this intervention scenario and its importance.

As discussed in Chapter 3, in the distance-1 scenario, intervention is applied to people who have a certain fraction of infected neighbors. More specifically, a person voluntarily chooses to take a vaccine if the number of infectious neighbors around him exceeds a certain threshold. Hence, the decision to intervene an individual depends on the health state of his direct contacts. Performing this intervention involves operating on large social contact networks and computation of people-people interaction. As a result, this intervention scenario becomes complex because it is both data and compute intensive. Simulating this scenario requires high CPU and memory resources. Therefore, it becomes practically impossible to simulated this scenario on a standalone database system with limited resources. In fact, as per the results in Table 8.3, it was impossible to perform the distance-1 study on Oracle for larger datasets such as Chicago and LA.

However, at the same time, this is an important intervention scenario and has been studied [30]. In the case of a raging epidemic, a close social contact network with a lot of interactions can aggravate the disease transmission. In such networks, if the neighbors of an individual do not get vaccinated, the disease will virulently spread from person-to-person.

Using PostgresXC, we were able to simulate the distance-1 in about 37 minutes for one of the largest datasets - LA. Figures 8.6 and 8.7 show the execution time performance of PostgresXC in simulating the distance-1 intervention over three regions - Miami, Chicago and LA.

The first figure 8.6 shows the results on the fat compute node(Singlenode-Multicore setup) and the second figure 8.7 shows the results on the distributed supercomputer nodes (Multinode-Multicore setup). In the case of the fatnode run, the number of datanodes was varied from 1 through 32. In the distributed case, the number of datanodes was set to 2, 2 and 4 per compute node for the states Miami, Chicago and LA respectively.

From the two figures, we observe that the total simulation time, bulk of which is the database access time, scales well on both the setups and for all the datasets.

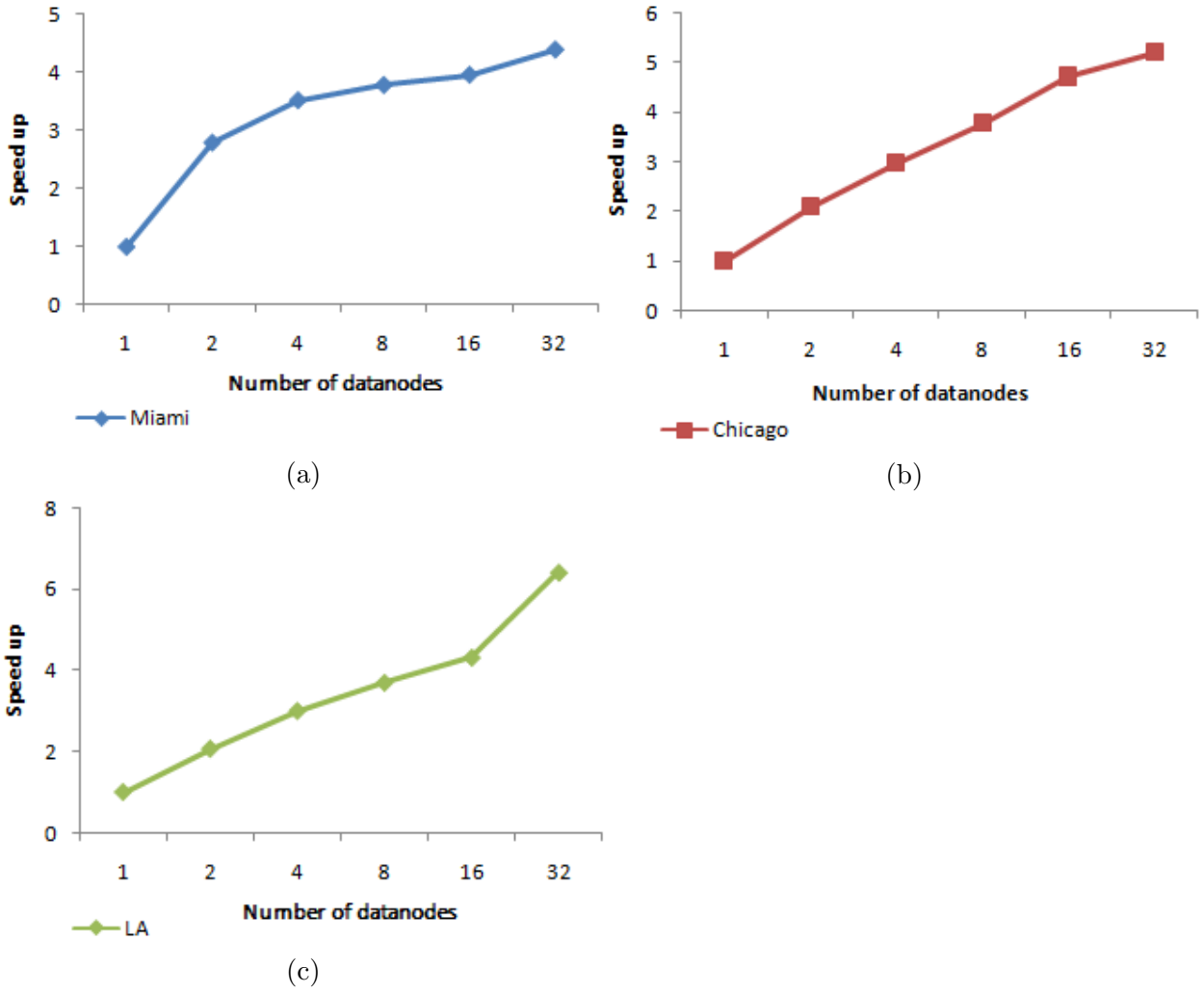


Figure 8.6: Characterizing the scalability of IndemicsXC for the distance-1 intervention on Singlenode-Multicore setup i.e., the fat node. The plot summarizes how IndemicsXC scales with the number of datanodes for a fixed-size contact network. (a) Strong scaling performance over Miami. (b) Strong scaling performance over Chicago. (c) Strong scaling performance over LA. The three plots show that as the data per datanode decrease (and the number of datanodes increase), there is a reduction in the overall query time.

8.2.2 Weak Scaling

Another way to measure the performance of IndemicsXC is through weak scaling, in which contact network size is increased as the number of datanodes increase. The idea is to complete the execution of a bigger dataset(contact network) in the same time as the smaller dataset by adding more resources(datanodes). For this experiment, we selected four U.S. areas with increasing population sizes: Miami, Boston, Chicago and LA. The contact networks of these areas have similar average degrees. The intervention applied in this experiment is distance-1,

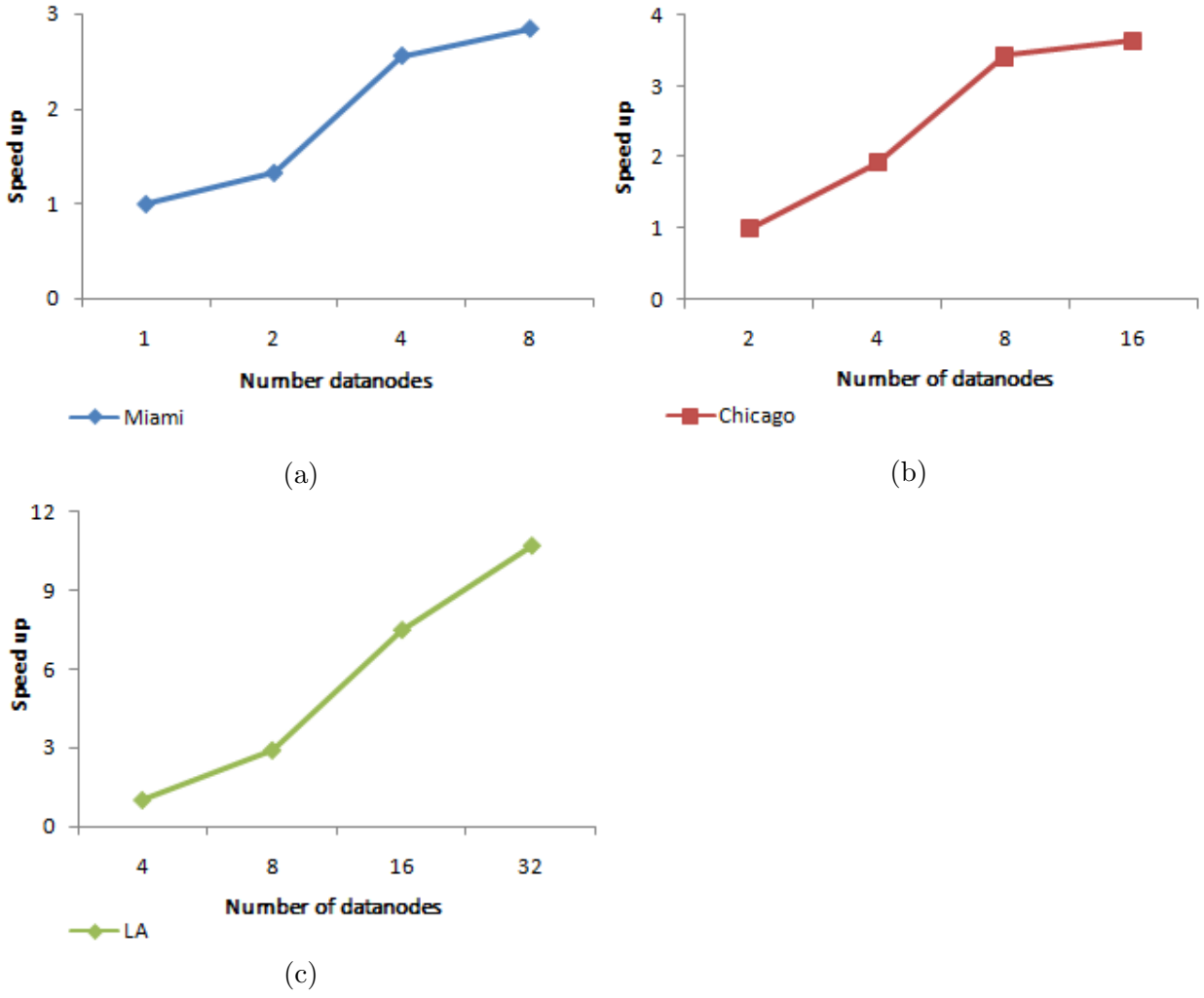


Figure 8.7: Characterizing the scalability of PostgresXC for the distance-1 intervention on Multinode-Multicore setup i.e., the distributed cluster nodes. The plot summarizes how IndemicsXC scales with the number of datanodes for a fixed-size contact network. (a) Strong scaling performance over Miami. (b) Strong scaling performance over Chicago. (c) Strong scaling performance over LA. Same as in case of fatnode run, we observe increase in parallel speedup with the addition of more datanodes.

where an individual voluntarily takes vaccine if a fraction of his neighbors is diagnosed. As shown in Figure 8.8, the parallel speedup increases with the number of datanodes. This increase in the number of datanodes is in proportion to the dataset size.

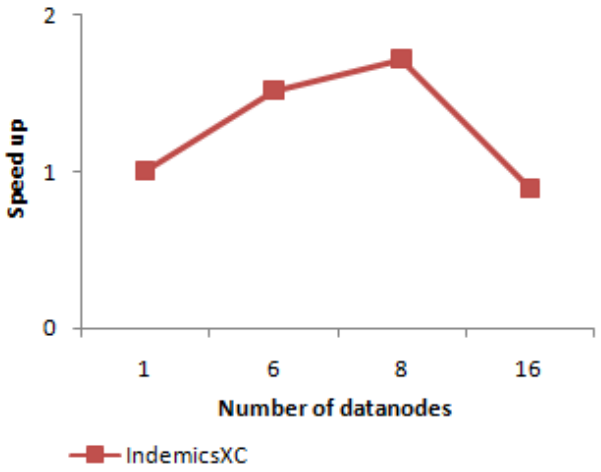


Figure 8.8: Characterizing the scalability of IndemicsXC for the distance-1 intervention on Singlenode-Multicore setup i.e., the fat node. The plot summarizes how IndemicsXC scales with the number of datanodes for varying size of social contact network. The curves show the speedup when the number of datanodes are added in proportion to the contact network size. For instance, Chicago contact network is 7.2 times the size of Miami contact network, hence the number of datanodes used in case of Chicago will be 7.2 times as many as in Miami.

8.2.3 Effect of Intervention Trigger parameter on the query performance

Trigger, in an intervention scenario, is an event that activates an intervention. When triggered, the intervention is applied to the selected subpopulation(or the whole population). In all our studies, trigger is a set threshold value which when exceeded, will activate an intervention. In the school study, a school is closed if the number of diagnosed students exceed 5%. A higher threshold value causes increased number of infections and thus increased data read/write operations in the database. Therefore, a higher threshold value will increase the overall query processing time. Figure 8.9 shows that indeed a run with a lower threshold performs better than the one with a higher threshold value.

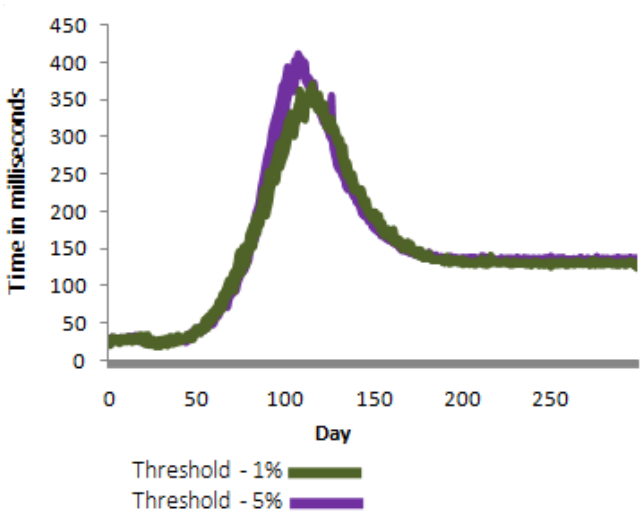


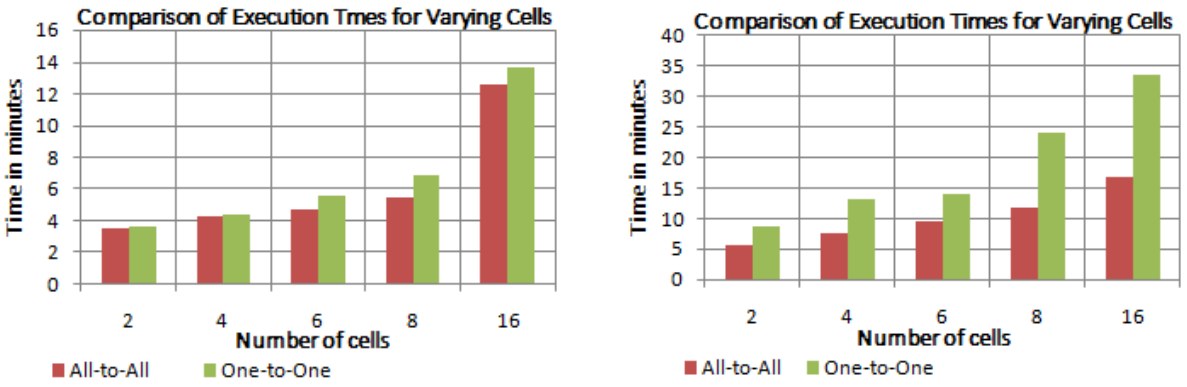
Figure 8.9: School Closure interventions: effect of threshold value on the query performance. The curves show the average running time of the database queries for threshold values - 1% and 5%. The width of curves represents one standard deviation above and below the mean. The run was simulated for 300 days with 5 replicates over Chicago region.

8.3 Evaluation of IndemicsXC Task/Data Distribution Strategies

8.3.1 One-to-One vs. All-to-All : Varying cells

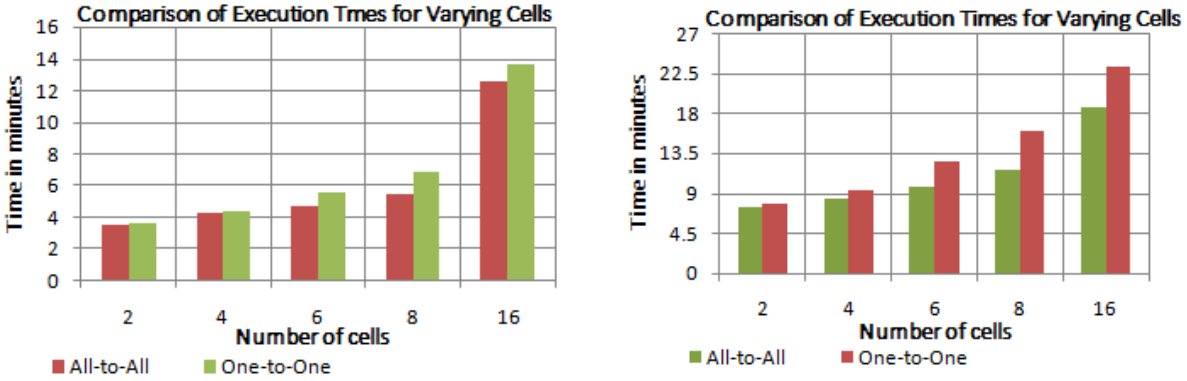
This experiment shows a comparison of the two execution strategies: one-to-one and all-to-all, but with varying number of simulation cells that are run concurrently. In this experiment (see Figures 8.10 and 8.11, all experiment parameters are same as the default configuration with the exception that the number of datanodes are fixed, and number of simulation cells are varied from 1 to 16. We instantiated datanodes on two compute nodes with four datanodes per compute node. The study scenario simulated are block and school interventions over Miami and Chicago region.

The above plots show better overall performance for all-to-all strategy as compared to one-to-one strategy. It is observed that for a small number of cells, both the strategies seem to perform similar. However, as the number of cells increase, the distributed approach (all-to-all) outperforms the replication strategy (one-to-one). Hence, for the given dataset and intervention scenario, we observe better throughput with all-to-all strategy.



(a) (b)

Figure 8.10: Performance comparison between the two High Throughput strategies for the block intervention and with varying number of parallel simulation cells. The plot shows the total execution time of the simulation for the two strategies over Miami and Chicago region. It can be seen that all-to-all distribution is better suited for the given intervention scenario. With the same amount of resources, all-to-all strategy results in better throughput. It performs 2X better than one-to-one for a run with 8 parallel cells over Chicago region.



(a) (b)

Figure 8.11: Performance comparison between the two High Throughput strategies for the school intervention and with varying number of parallel simulation cells. The plot shows the total execution time of the simulation for the two strategies over Miami and Chicago region. It can be seen that all-to-all distribution is better suited for the given intervention scenario. With the same amount of resources, all-to-all strategy results in better throughput. It performs 1.37X better than one-to-one for a run with 8 parallel cells over Chicago region.

8.3.2 One-to-One vs. All-to-All : Varying datanodes per compute node

This experiment also compares the two execution strategies but with varying number of datanodes. For this experiment (see Figure 8.12), we instantiated datanodes on two compute

nodes and varied the number of datanodes (per compute node) from 1 to 6. The other parameters are same as the default setting. The study scenario is Block intervention over Chicago region with four cells.

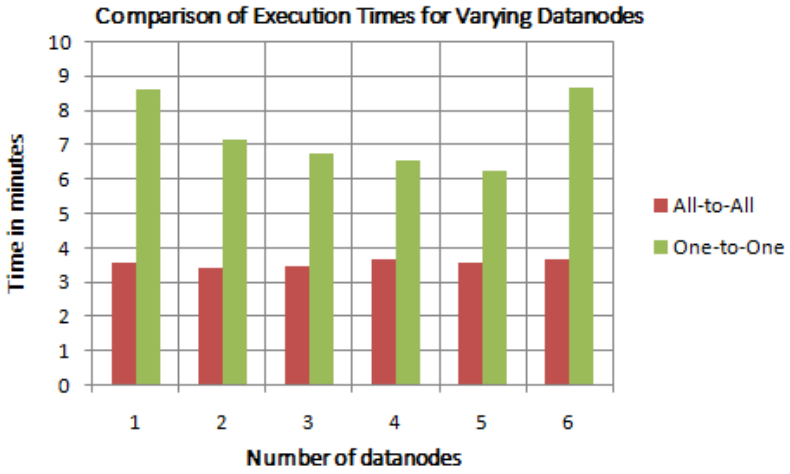


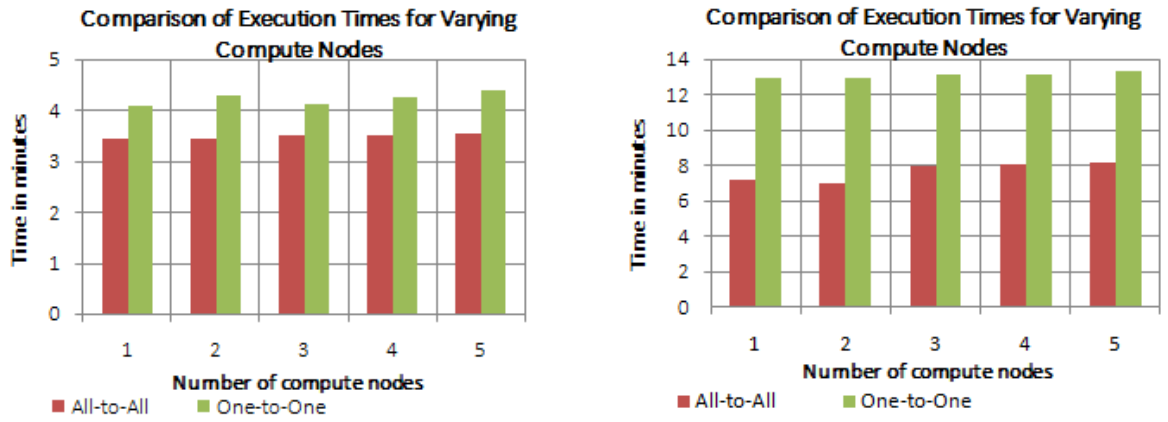
Figure 8.12: Performance comparison between the two High Throughput strategies for the block intervention and with varying number of datanodes. The plot shows the total execution time of the simulation for the two strategies over Chicago region and block as the intervention scenario. Note that all-to-all again performs better than one-to-one approach in this scenario.

The results suggest that adding datanodes improve the query performance as it enhances parallelism as is evident from the graph. However, it can be clearly seen that after a certain point, adding more datanodes does not have any effect on the performance, rather in case of Chicago study, it seems to have degraded the query performance. This *threshold point* is decided by the CPU cores on a given compute node. Since ours is a 6 core system and hence, the optimal performance is expected for < 6 datanodes.

8.3.3 One-to-One vs. All-to-All : Varying compute nodes

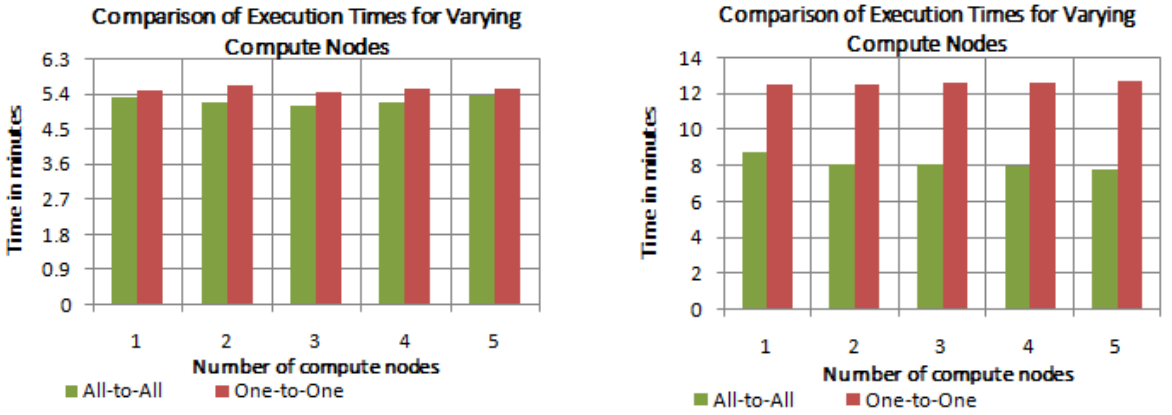
This experiment shows a comparison of the two execution strategies: one-to-one and all-to-all but with varying number of supercomputer compute nodes. In this experiment (see Figure 8.13), we instantiated datanodes on varying compute nodes-four datanodes per compute node. The other parameters are same as the default setting. The study scenario simulated is Block intervention over Miami and Chicago region with four cells.

We did a similar experiment for School intervention (see Figure 8.14) by varying compute nodes from 1 to 5. The above results reveal that simulations scale with the number of datanodes per compute nodes. However, the plots also indicate that adding compute nodes do not necessarily improve the performance as is evident from the spike in the two graphs after adding 4 compute nodes. It can be seen that after a certain point, adding more compute



(a) (b)

Figure 8.13: Performance comparison between the two High throughput strategies for the block intervention and with varying number of compute nodes. The plot shows the total execution time of the simulation for the two strategies over Miami and Chicago regions. It is observed that for the same amount of resources for a given run, all-to-all strategy performs better. However, there seems to be a degradation in the execution times after increasing the number of datanodes beyond two. This degradation can be attributed to increased overhead at the coordinator.



(a) (b)

Figure 8.14: Performance comparison between the two High throughput strategies for school intervention and with varying number of compute nodes. The plot shows the total execution time of the simulation for the two strategies over Miami and Chicago regions. It is observed that for the same amount of resources for a given run, all-to-all strategy performs better. However, there seems to be a degradation in the execution times after increasing the number of datanodes beyond two. This degradation can be attributed to increased overhead at the coordinator.

nodes do not have any effect on the performance of either strategy (all-to-all and one-to-one). It could be because the amount of computation being done at the four compute nodes is less than the overhead of network communication. The network communication overhead due to the added compute node could have been more than the amount of computation done at the datanodes. Hence, even though the additional compute node parallelizes part of the computation, the network overhead it introduces consumes more power than the computation itself.

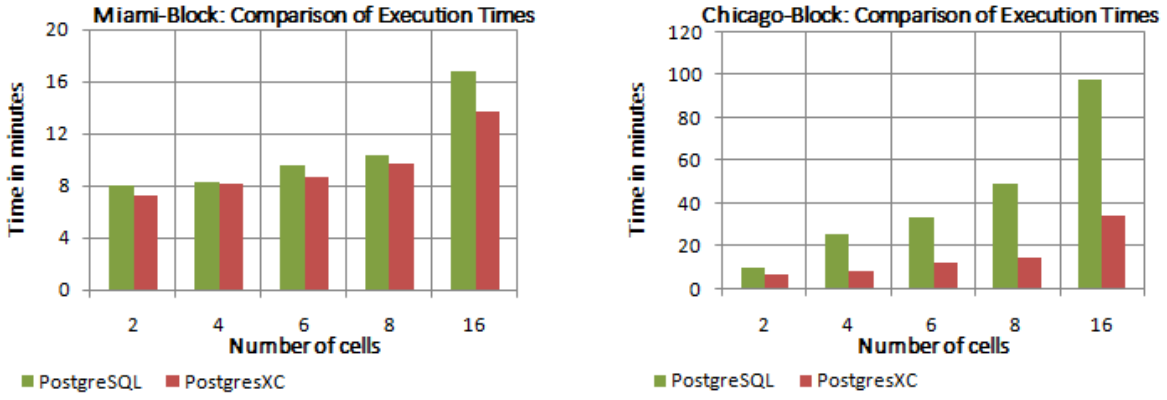
8.3.4 Throughput comparison : Multinode-Multicore (scaling up) and Singlenode-Multicore (scaling out)

We analyzed the network overheads of the communication between datanodes and the coordinator in a distributed environment in which datanodes are distributed across several physical machines. For this analysis, we did an experiment on a *fatnode* machine (Multinode-Multicore) and instantiated the datanodes on a single machine. We compared the results from the fatnode run with the distributed experiment (Singlenode-Multicore) over Chicago region simulating school intervention (see Figure 8.15). The figure shows that for the given simulation scenario and dataset, communication latency between the coordinator and the datanodes does not seem to have any significant impact on the performance. Rather the parallelism offered by distributing the datanodes outweighs any network overhead.

Figure 8.15: Performance comparison of PostgresXC - on a Single Fatnode and Distributed cluster nodes. The plot gives the total running time of Indemics with PostgresXC as the RDBMS. The results are for Block intervention scenario over Chicago region. The number of datanodes are varied from 1 through 16.

8.3.5 Throughput comparison: Standalone (Indemics on Postgres) vs. Distributed database

Figures 8.16a and 8.16b present a performance comparison between PostgresXC in IndemicsXC and standalone PostgresSQL for block intervention over Miami and Chicago. The experiments were conducted on a single compute node and with varying number of simulation cells. The number of datanodes for the distributed database run have been set to 2 for Miami and 6 for Chicago. All other parameters are same as the default. Experiments indicate that using a distributed database improves the intervention performance considerably. The results over Miami region show that the distributed database gives a slightly better performance than standalone as the number of simulation cells increase. The performance improvement is much more apparent for the Chicago dataset with much better simulation time results.



(a) (b)

Figure 8.16: Performance comparison between Distributed PostgresXC and Standalone PostgreSQL in supporting multiple parallel simulations. PostgresXC is instantiated on a single compute node with 2 datanodes for Miami and 6 datanodes for Chicago. The execution difference is more in the Chicago run than Miami. As seen from the plot 8.5, the time spent in database access is more in Chicago than Miami, hence the bigger difference.

8.3.6 Data Distribution - Replication vs. Distribution

In this section, we study the impact of data distribution on the query performance. As discussed in Chapter 7.1, choosing the right data distribution strategy is critical for the query performance. We present the results from the school study to illustrate this point. These experiments cover two scenarios- in the first case student table is replicated on all datanodes and in the second case, student table is distributed across the datanodes. Replicating the table, in this case results in pushdown of the join and aggregate operation in an intervention

query, while distribution results on pushdown of the join and partial pushdown of the aggregate operation. It is expected that the case where the entire query gets pushdown yields the better result. We simulated the run over Miami and Chicago regions and varied the number of simulation cells. Figure 8.17 shows the ratio of the execution times of replication to the distribution strategy. These studies show that in the distribution strategy in which a join is not pushed down leads to a large amount of data movements and severely affects the performance. Hence, the objective of our distribution strategy is to push down joins and minimizes data movement. This can be achieved either by distributing both the tables on the join column or replicating one of the tables. We have studied both the scenarios and found that replication works better than distribution in this case.

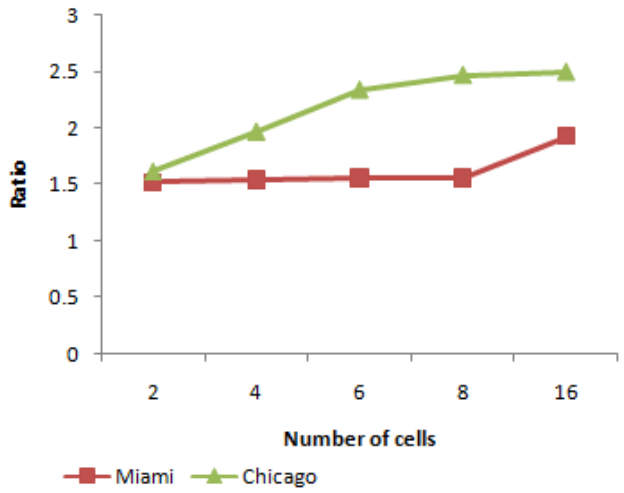


Figure 8.17: Performance comparison between two data distribution strategies - Replication and Distribution. There are two cases- one in which the student network table is replicated and second in which the student table is distributed. The default work distribution strategy is all-to-all. The two plots show the ratio of total execution time of these two strategies for Miami and Chicago networks in school intervention. In this intervention scenario, replicating the table results in join and aggregate pushdown, while as distributing the table results in join and partial aggregate pushdown. From the results, we see that the replication technique works better than distribution in the given case.

8.4 Evaluation of different optimization techniques in IndemicsXC

Performance improvement due to optimization techniques - query pushdown and table partitioning:
We now evaluate the effectiveness of the optimization techniques discussed in Chapter 7. We measure the performance of our optimization strategies for some selected intervention

scenarios and datasets and compare it against their traditional implementations. The results are summarized in Table 8.4. It can be observed that we gain good overall speedup by incorporating the optimization strategies in the intervention computation.

Table 8.4: Evaluation of different intervention query optimization techniques. Performance improvement is shown in terms of speedups over the traditional query processing techniques.

Region	**Intervention study**	**Optimization technique**	**Performance improvement**
Miami	Block	Copy operation	8.3X
Miami	Block	Inserts pushdown	12.5X
Miami	Block	Updates pushdown	22X
LA	D1	Simulation table partition	4X

8.4.1 Cost of query planning

As described earlier in Chapter 6.4, an intervention query goes through several phases of execution. One of the initial phases is query planning. For every query, multiple query plans are generated by the planner. Out of all these generated plans, the most optimal plan is chosen by the optimizer for execution. Several technical and physical factors affect the quality of plan chosen such as the system related configuration parameters, planner cost constants, running services for regular cleanup of deleted/ updated records, etc. The cost of query planning may have an impact on the overall execution time. The table below 8.5 illustrate the average query planning time for the three interventions: block, school and distance-1 over Chicago region. The query planning and execution costs for each intervention shows the average costs of all the queries (see Chapter3 for reference). In our case, we found that the query planning cost is negligible.

Table 8.5: Comparison between average query planning and overall execution cost for the given interventions over Chicago dataset.

Intervention	**Average Query planning cost (In milliseconds)**	**Average execution cost (In milliseconds)**
Block	1.502	58.96
School	1.22	348.89
Distance-1	1.31	4852

8.4.2 Indexing:

Creating indexes on the static tables facilitates fast data look-up. Different types of indexes supported by PGXC are B-tree, GiST, GIN, etc. For all our studies - block, school and honest, we created B-tree indexes on all the static tables. For some of the static tables such as BlkPop and Student, we created multiple indexes on different table columns. These columns are typically referenced in the *where* or a *join* clause of the queries. To show that the intervention queries benefited from the indexes, we did a comparison of queries with and without indexes on the static tables. This experiment was conducted with default configuration settings. The results from Figure 8.18 show that having indexes improve the performance of the intervention queries.

Figure 8.18: Comparison of performance of a sample query from Block intervention with and without indexes on all the tables. The runs were simulated over Miami region. This difference in the execution times is expected to increase with the size of the datasets.

Chapter 9

Cost Model

9.1 Cost model for intervention queries

Estimating the performance of an intervention under different database strategies is an important milestone in planning and designing an experiment and also in deciding the allocation of resources i.e. the number of datanodes. Given a simulation scenario and a dataset, we should be able to find a database configuration which would yield the maximum performance. For example, the database configurations for simulating interventions over Miami and LA would not be same because of a huge difference in the data-size. Similarly, for running a complex intervention like Distance-1, the setup would be different from running a much simpler Block study. To that end, we present a performance model that estimates the performance of IndemicsXC. The model predicts the best database configuration based on total query cost for a given population and intervention for an approximate size of the infections per day. The model computes the cost of each query operator in an intervention query and based on that decides the performance for given configuration. The cost model will come up with an optimal strategy for simulating an intervention and would help in maximizing its performance in terms of the throughput and execution times. Our model extends the work on the performance model for intervention queries by Ma et al. [24]. They modeled the Indemics performance based on atomic query operations such as *join, aggregate, insert,* etc. Their work is based on results obtained from a standalone database. They profile the query performance from the prior runs and maintain a performance library which can be looked up to know the performance of different query operations under different configurations. However, in contrast to their work, we don't maintain a pre-compiled list of query performance, rather we approximate the values using data from regression analysis. Our model has a set of parameters that includes information about dataset and approximate infections per day. From those parameters, we predict the query performance and whether a given configuration is optimal.

Interventions are computed on the database side and are supplied to the simulation engine

via the Indemics server. An Intervention has a type that specifies the action along with its parameters and a sub-population to be intervened. The population to be intervened gets calculated dynamically within the database as the simulation progresses. These interventions (as explained in chapter 1.1.1) are SQL-like statements that are written in a client script for batch submission. At runtime, intervention statements in the client script get translated into database queries and are executed for each day of simulation. In the client script, at each time-step, sub-population to be intervened gets chosen and at the last statement, the final sub-population to be intervened for that particular day gets selected.

We have taken queries from the Distance-1 study described in the chapter 3.3.1. These queries have been broken down into basic query operations and classified in the Table 9.1.

Table 9.1: Atomic query operations in involved in the intervention queries.

Operation	**Notation**	**Sample query**
Index scan	γ	index scan on $R(\alpha)$
Projection	π	select α from R
Predicate	θ	$\alpha = x$
Selection	σ	select $*$ from R *where* $\alpha = x$
Insert	ω	insert into R *from* X
Copy Out	Ω_{out}	copy R *to* $< file >$
Copy In	Ω_{in}	copy R *from* $< file >$
Update	Θ	update R *set* $\alpha = x$
Natural Join	$\bowtie$	select $*$ from R, S *where* $R.\alpha$=$S.\alpha$
Group By	Σ	group by α

In our performance model, in addition to above atomic query operations, there are other costs involved such as 1) data transfer cost between coordinator and the datanodes. 2) aggregation of results at the coordinator from all datanodes. The data transfer and aggregation costs are represented as ψ and $\sum$ respectively. Table 9.2 lists some additional notations used in the performance model.

Table 9.2: Additional notations used in the performance model

Operation	**Notation**	**Description**
Data transfer	ψ	Transfer of results from datanodes to the coordinator.
Parallel operation	$\prod(O)$	parallel processing of an operation O
Aggregation	$\sum$	Aggregation of results at the coordinator.

9.1.1 Cost of atomic operations

We now derive the costs for queries that involve the above-described atomic query operations. Let Q be the number of processors per compute node. Let C denote the coordinator or the master node. and D be the number of datanodes per compute node. In all cases, $D \leq Q$. Let dn be the total datanodes. Now, consider the following example, Let $R(a, b)$ and $S(c, d)$ be the two relations we want to use in our model. $R(a, b)$ and $S(c, d)$ represent the static and dynamic tables used in a simulation. Let R and S be hash distributed on attributes a and c respectively. The two tables are hash distributed across dn datanodes. The cost of the atomic operations involved in the Distance-1 intervention scenario is given below:

1. **Select with predicate** : $\sigma_\theta R$, where predicate θ is $a = x$, a being the partitioning attribute.
   $\prod\left(\sigma_\theta \frac{|R|}{dn} \times \gamma_R\right)$ + Data transfer + Aggregation cost
   Data transfer cost is the for sending the results from the datanodes to the coordinator. Aggregation cost is the cost at the coordinator for aggregating the output from all datanodes

2. **Select with predicate and a join between relations R and S**:
   $\prod\left(\sigma_\theta \frac{|R|}{dn} \times \gamma_R\left(\frac{|S|}{dn} \times \gamma_S\right)\right)$ + Data transfer + Aggregation cost

3. **Insert** : $\omega_x R$, where x denotes the rows to be inserted.
   $\prod\left(\omega_x \frac{|R|}{dn}\right)$

4. **Copy** : $\Omega_x R$, where x denotes the rows to be copied to/from the relation R.
   $\prod\left(\Omega_1 \frac{|R|}{dn} \times x\right)$.

5. **Update** : $\Theta_{\alpha=x} R$, where α is the attribute to be updated with value x.
   $\prod\left(\Theta_{\alpha=1} \frac{|R|}{dn} \times x\right)$.

6. **Group By** : $\Sigma_\alpha R$, where α is the attribute of relation R on which a Group By is to be performed.
   $\prod\left(\Sigma_\alpha \frac{|R|}{dn}\right)$ represents a Group By at the datanodes.

7. **Data transfer** : ψ denotes the cost incurred in transferring the output from the datanode(s) to the coordinator.
   $\psi = \sum \lambda_{j=1}^{dn}$
   here $\lambda = \delta \times x$ where δ is the cost of transferring one row to coordinator from a datanode.

9.1.2 Contributing factors in estimating performance

In this section, we highlight four important factors that affect the estimation of the performance. They are listed below:

1. Caching effect : In the case when the entire table could not fit into the memory and the process goes back and forth to fetch the rows from the disk, caching has an important role. The more the data is cached; the better is the performance.

2. Number of processors : This limits the degree of parallelism that we can get by running queries against several datanodes. For example, a compute node with 6 cores would not benefit much even if we increase the number of datanodes to say 16.

3. Degree of parallelism: This is the parallelism achieved by running a query operator in parallel on many datanodes. To some extent, this depends on the way coordinator is designed to do a particular query operation. For example, for doing Copy operation on several datanodes, coordinator lets the copy stream to be connected to datanodes. Hence, copy can be done in parallel on the datanodes. However, in case of such operations as inserts, the coordinator scans each row and then does an insert. Hence, using partitioned data across multiple datanodes won't help much in this case.

4. Degree of selectivity: This has an impact on the data transfer cost from the datanodes. High selectivity means fewer output rows from datanodes that mean less work done by the coordinator in aggregation. At the same time, low selectivity could mean more data transfer cost as the coordinator receives the data serially from the datanodes.

9.1.3 Assumptions in the model

Table 9.3 lists the assumptions about the algorithms used in various operations involved in the intervention queries. These operations are: scan, join and aggregate. In the table, n is the number of input rows to a query operator. The hash aggregate includes the cost of building a hash table and also probing it for aggregation.

Table 9.3: List of assumptions in the performance model

Type of Operation	**Assumed algorithm**	**Complexity**
Scan	index scan	$\mathrm{O}(n)$
join	nested loop	$\mathrm{O}(n^2)$
Aggregation	hash aggregate	$\mathrm{O}(n)$

9.1.4 Performance estimation

For a given query, the performance model can be helpful in determining the right distribution strategy with given resources. It also helps to decide the number of resources that must be allocated to a given experiment. The performance model will determine the best strategy for a given query. We will evaluate the correctness of our performance model by validating its estimated (theoretical) time with actual (experimental) time.

We will compare the model results with the experimental results described in Chapter 8. This will validate the results from the performance model with the experimental data. As explained in Chapter 9.1, the intervention statements get translated into a database query at runtime. The total query cost would include the actual query processing at the datanodes plus the cost of sending the data back to the coordinator node. This can be denoted as:

$$\text{Total Cost query } = \text{Query processing } + \text{ Data transfer}$$

Let us now consider sample queries from Distance-1 scenario (see Chapter 3.3) and model the performance of the entire study. The queries in the given scenario get executed per day, at the end of which the id's of the intervened people are selected. We first calculate the individual cost of the each query by computing the cost of an atomic operation involved in it, followed by aggregated cost per day.

Let,
nf be the neighbor factor which gives the average number of neighbors per person,
dc be the number of diagnosed cases per day for the given intervention study,
df be the damp factor which gives the average common neighbors in the given social contact network,
C_i be the cost of i^{th} query per day,
R_i be the result set for i^{th} query and
D be the total simulated duration and d be the current day.

The cost of each query from the distance-1 intervention is computed below:

1. **Copy to *Diagnosed_Count***: people diagnosed on the current day.

   $$C_1 = \prod \left(\Omega_1 \times \frac{dc}{dn} \right)$$

   $$R_1 = dc(\frac{dc}{dn} \text{ per node})$$

2. **Copy to *D1DiagDailyCount* :** get the count of infected neighbors of a person on

the current day.

$C_2 =$ index scan cost + group by aggregate cost + aggregate cost at the coordinator + data transfer cost + insert cost

$$C_2 = \prod((\gamma + \frac{dc}{dn} \times \gamma) + \Sigma_{tail}(\frac{dc}{dn} \times df \times nf) + (2 \times \Omega_1 \times \frac{dc}{dn} \times df \times nf)) + \sum \frac{dc}{dn} \times df \times nf + \lambda(\frac{dc}{dn} \times df \times nf) \times dn)$$

$$R_2 = dc \times df \times nf$$

3. **Insert into *D1InfWinAgg* :** gets the count of infected neighbors of a person for the last three days.

$C_3 =$ index scan cost + group by aggregate cost

$$C_3 = \prod(\gamma(3 \times \frac{dc \times df \times nf}{dn}) + \Sigma_{pid}(3 \times \frac{dc \times df \times nf}{dn}))$$

$$R_3 = 3 \times dc \times df \times nf$$

4. **Update *D1IntvDaily*:** intervene a person if the number of infected neighbors around him exceed a fixed threshold.

$C_4 =$ index scan 1 cost + index scan 2 cost + update cost

$$C_4 = \prod(\gamma + (3 \times \frac{dc \times df \times nf}{dn} \times \gamma) + \gamma + 3 \times \frac{dc \times df \times nf}{dn} \times \gamma + \Theta_1 \times 3 \times \frac{dc \times df \times nf}{dn} \times 0.03)$$

$$R_4 = 3 \times dc \times df \times nf \times 0.03$$

5. **Insert into Intervention table.**

$C_5 =$ sequential scan cost + insert cost

$$C_5 = \prod(\gamma + (\omega_1 \times 3 \times \frac{dc \times df \times nf}{dn} \times 0.03)$$

$$R_5 = 3 \times dc \times df \times nf \times 0.03$$

The total cost of an intervention per day would be the sum of individual costs of the queries.

$$Cost_d = C_1 + C_2 + C_3 + C_4 + C_5 \tag{9.1}$$

The total cost of the intervention for the simulated number of days can be given as:

$$Cost_{total} = \sum_{i=1}^{D} Cost_i, \text{ where } D \text{ is the total simulation duration.} \tag{9.2}$$

9.2 Model evaluation

In this section, we evaluate the effectiveness and accuracy of our proposed performance model by calibrating it with the results from the LA dataset. To predict the total query cost for the Distance-1 study, we need to substitute the values in equation 9.2. From previous Distance-1 study runs on 36 datanodes, we got an approximate data for infections per day. We computed the values of other model parameters from the LA dataset which are given in Table 9.4.

Table 9.4: Cost estimates for model parameters for distance-1 study and LA dataset.

Parameter	**Approximate value**
nf	50.37
df	50.37

We profiled the performance of different query operations mentioned in the Table 9.1 and approximated their cost through regression. Below Table 9.5 gives the comparison of approximate and actual cost of the five intervention queries described above on the 50^{th} simulation day.

Table 9.5: Cost estimates for the intervention queries from distance-1 study. The estimates are calculated for the 50th day of simulation. The five queries and their cost equations described in the previous section are numbered from 1 through 5 in the table.

Query	**Actual cost in milliseconds**	**Approximate cost in milliseconds**
1	317	236
2	80.634	63.8
3	451	411.747
4	53.89	39.578
5	74.89	72.05

The accuracy of the model can be improved by doing regression analysis of more experiments and probably including system parameters (caching, i/o) as well. Similarly, we can calculate the approximate cost of the intervention queries for the total duration of the simulation and get the approximate cost estimates.

We select the best configuration for a given intervention by running it through the cost model which, based on the performance parameters, determines how many nodes and datanodes should be used for a given experiment. In our model, data transfer cost function is an important parameter in deciding the right configuration. It is given by:

$$\text{Data transfer cost} = \frac{\text{width of all projected columns in bytes } \times \text{ number of projected rows} \times 8}{1500 \times 0.007}$$

where, 1500 is the MTU (maximum transmission unit) size and 0.007 is the time to transfer 1 MTU across the nodes in our supercomputer cluster. The transfer time increases as the task granularity decreases. It could mean there might not be any benefit by adding more nodes.

Chapter 10

Discussion

10.1 Why Parallel/Distributed Databases?

The motivation behind transition to distributed databases is based on the premise that standalone systems with limited resources can no longer meet requirements for cost-effective scalability and performance. In the large-scale simulations with complicated intervention scenarios, massive read/write operations happen. Hence, standalone systems may not be suitable for such applications. There are many frameworks and cluster database solutions such as Oracle RAC, MySQL cluster, Greenplum, Postgres-XC, and Hadoop to name a few. These systems can be broadly classified into: shared-disk(Oracle RAC), shared-nothing (MySQL cluster, Greenplum, Postgres-XC) and MapReduce(Hadoop). Oracle RAC is based on share-everything architecture, and that is where the problem lies. Because of a single shared-disk, there is limited IO processing power which also puts a limit on the expansion of capacity. Thus, even though the tables are partitioned across different servers, they all need to access the same disk.

Greenplum is a parallel database with distributed query processing built on top of PostgreSQL. It is a column store database which means that it stores columns of data together. Our group has done a preliminary study on the performance of Greenplum for running interventions. The results show that for complex interventions such as distance-1, PostgresXC performs better than Greenplum(about 1.54X better for LA region). Since, Greenplum is a proprietary database, there are limitations to the tuning that can be done to get the maximum performance benefit.

MySQL cluster is another high-availability, shared-nothing cluster DB, which with read and write scalability. Even though it offers query localization to pushdown JOIN operations, there are some serious limitations with table access with no primary key. There are no such restrictions in Postgres-XC.

Hadoop Map/Reduce is an open-source, java-based framework that offers parallel processing

of vast amounts of data. In the map phase, data is split into chunks and mapped to "map tasks." The output from the map tasks is sorted and fed to the "reduce tasks." The reason we believe that Hadoop might not be suited for driving intervention scenarios is because of the type of tasks involved in intervention computation. There are interleaved read/write operations involved in an intervention query which might be split into two separate "map" and "reduce" tasks. Thus, it is highly likely that this might be an overhead in case of complex interventions.

Therefore, we believe that Postgres-XC is most suited for dealing with the kind of workload involved in simulations.

10.2 Considerations For Using Distributed Databases For Driving Interventions

Performance and scalability have become a wide-split concern today. Scalability is desired in everything, from a small web application up to large scientific simulations or even database. The ever increasing size of datasets in various fields necessitates the efficient data management and processing techniques. Theoretically, a distributed database as back-end for a large, complex simulation would look like a good solution to the address scalability issues. One might think that simply adding more resources and power is enough to improve the performance. While it may be true in some cases, for instance, web servers, several embarrassingly parallel applications, distributed rendering in computer graphics, etc., addressing the problem using distributed databases is still a non-trivial solution. In this chapter, we discuss the scalability solution offered by distributed databases and whether all HPC based simulations would benefit from using it as back-end. We base our discussion on three key points: 1) Right data partitioning: the importance of choosing an effective way of distributing the data across the datanodes. 2) Task granularity: efficient distribution of task among datanodes such that the ratio of computation to the amount of computation is maintained. 3) Reliability vs. Performance: we can get very good performance from the distributed databases when we turn off the reliability parameters. Since reliability comes at a price of performance, one has to make a conscious choice to select one of the two.

A critical factor for scalability and performance is the effective partition of large datasets in order to prevent skewed data. One of the factors that decide the **efficient partitioning** is the data distributing technique we choose. There are three data distribution strategies provided by PostgresXC - Hash, RoundRobin and Modulo. If the data column on which we decide to distribute the table has all unique values, then hash or modulo strategy would probably distribute the data evenly. If, however, the data values in the distribution column are not unique or have multiple duplicate values then in that case, it would be prudent to use the RoundRobin strategy which always distributes the data evenly. But RoundRobin strategy does not always push down queries (for reference see 6.5). For instance, given the

three distribution strategies, if we have a query that involves a join between two large tables, then which should we choose. In this case, we have to decide which of the two- skewed data or serial query processing would pose a bigger issue. That too depends on a lot of factors such as the size of data/result set, query complexity, whether it involves data modification operations like updates and inserts which are almost always expensive when done serially. So, our first consideration is the effective data partitioning for optimal performance.

A second important factor which impacts the scalability is the **task granularity**, i.e., to how much extent should we divide the data. In terms of a distributed database, it means, how many datanodes should be instantiated for holding a fraction of the distributed data. It also important to know if it is required to distribute the data among multiple datanodes. For instance, if a given dataset is so small that distributing it across multiple nodes does not improve the performance, rather degrades it, then we need to re-think out decision of using a distributed database. So our second consideration is the **size of datasets** and the result set. If the result set is so small that the time to compute it takes substantially less time to sent it across the network then we will low computation to communication ratio. It might degrade the performance rather than improving it.

Another important question we can ask is: "which of the two is more important for my application - **reliability or performance**?". If our answer is former, then we might want to think if distributed database is the solution to our scalability problem. The distributed systems are prone to multiple points of failures. Even HPC systems like supercomputers are not immune to node failures. For our application, performance is the main criteria. In running epidemiological studies, reliability is not of the utmost importance. While the simulation is running on an HPC system, and if there is a database node failure, we can immediately start over without much loss. It, however, is not the case for mission critical database applications which require scalable and continuous services. Hence, this may be one of the factors that decide the use of distributed databases like PostgresXC for reliable data access.

Chapter 11

Conclusion

Standalone databases don't have massive processing power or large memory resources to handle complex, realistic intervention scenarios in epidemiological simulations. We addressed this by problem of scaling the data-engine by using distributed databases on supercomputers. In this thesis, we mainly focused on enhancing the DBMS portion of Indemics using the high performance distributed database and also evaluated its performance by comparing it with traditional standalone databases. The distributed database offers extensive scalability by offering massively parallel query processing. The database being open source, we were able to extend the existing database and made simulation specific modifications and optimizations which enhanced the parallelism on the datanodes through the query pushdown. we also designed data distribution, execution strategy, and optimizations to the IndemicsXC system for scaling multiple simulations running in parallel. By leveraging the parallelism and scalability offered by distributed database, interventions queries were processed in parallel that enabled fast simulation, thus making the real-time pandemic planning more efficient. Finally, we presented a cost model for predicting the performance of queries from Block and Distance-1 interventions. The cost model also predicts the best configuration for simulating the given intervention scenario. Our optimizations and high throughput strategies are not restricted to Indemics and can be extended to any application that uses a database as a back-end for data and compute intensive tasks.

Bibliography

[1] Postgres-xc, 2013.

[2] Sanjay Agrawal, Vivek Narasayya, and Beverly Yang. Integrating vertical and horizontal partitioning into automated physical database design. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 359–370. ACM, 2004.

[3] Theodore T Allen. Introduction to discrete event simulation and agent-based modeling. *Columbus: Springer*, 2011.

[4] 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.

[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 *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 37. IEEE Press, 2008.

[6] Gordon Bell, Jim Gray, and Alex Szalay. Petascale computational systems. *Computer*, 39(1):110–112, 2006.

[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] Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences of the United States of America*, 99(Suppl 3):7280–7287, 2002.

[10] Zhuhua Cai, Zografoula Vagena, Luis Perez, Subramanian Arumugam, Peter J Haas, and Christopher Jermaine. Simulation of database-valued markov chains using simsql. In *Proceedings of the 2013 international conference on Management of data*, pages 637–648. ACM, 2013.

[11] IBM DB2. Partitioned tables, 2007.

[12] Mark Dredze. How social media will change public health. *Intelligent Systems, IEEE*, 27(4):81–84, 2012.

[13] E.Bonabeau. Building accurate predictive models without data, March 31 2014.

[14] Harvey V Fineberg and Mary Elizabeth Wilson. Epidemic science in real time. *Science*, 324(5930):987–987, 2009.

[15] Edward Givelberg, Alexander Szalay, Kalin Kanov, and Randal Burns. Mpi-db, a parallel database services software library for scientific computing. In *Recent Advances in the Message Passing Interface*, pages 339–341. Springer, 2011.

[16] Jim Gray, David T Liu, Maria Nieto-Santisteban, Alex Szalay, David J DeWitt, and Gerd Heber. Scientific data management in the coming decade. *ACM SIGMOD Record*, 34(4):34–41, 2005.

[17] Jim Gray, David T Liu, Maria Nieto-Santisteban, Alex Szalay, David J DeWitt, and Gerd Heber. Scientific data management in the coming decade. *ACM SIGMOD Record*, 34(4):34–41, 2005.

[18] Jim Gray and Alexander S Szalay. Where the rubber meets the sky: Bridging the gap between databases and science. *arXiv preprint cs/0502011*, 2005.

[19] Gerd Heber, Chris Pelkie, Andrew Dolgert, Jim Gray, and David Thompson. Supporting finite element analysis with a relational database backend; part iii: Opendx–where the numbers come alive. Technical report, Microsoft Research Technical Report, 2005.

[20] Herodotos Herodotou, Nedyalko Borisov, and Shivnath Babu. Query optimization techniques for partitioned tables. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 49–60. ACM, 2011.

[21] Herbert W Hethcote. The mathematics of infectious diseases. *SIAM review*, 42(4):599–653, 2000.

[22] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher Jermaine, and Peter J Haas. Mcdb: a monte carlo approach to managing uncertain data. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 687–700. ACM, 2008.

[23] Ebbing Lautenbach, Sanjay Saint, David K Henderson, and Anthony D Harris. Initial response of health care institutions to emergence of h1n1 influenza: experiences, obstacles, and perceived future needs. *Clinical infectious diseases*, 50(4):523–527, 2010.

[24] Yifei Ma, Keith R Bisset, Jiangzhuo Chen, Suruchi Deodhar, and Madhav V Marathe. Efficient implementation of complex interventions in large scale epidemic simulations. In *Winter Simulation Conference*, pages 1354–1366, 2011.

[25] David Maier and David M Hansen. Bambi meets godzilla: Object databases for scientific computing. In *Scientific and Statistical Database Management, 1994. Proceedings., Seventh International Working Conference on*, pages 176–184. IEEE, 1994.

[26] Richard C Murphy, Kyle B Wheeler, Brian W Barrett, and James A Ang. Introducing the graph 500. *Cray Users Group (CUG)*, 2010.

[27] Hiroshi Nishiura, Don Klinkenberg, Mick Roberts, and Johan AP Heesterbeek. Early epidemiological assessment of the virulence of emerging infectious diseases: a case study of an influenza pandemic. *PLoS One*, 4(8):e6852, 2009.

[28] M Tamer Özsu and Patrick Valduriez. *Principles of distributed database systems*. Springer, 2011.

[29] 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.

[30] Ana Perisic and Chris T Bauch. Social contact networks and disease eradicability under voluntary vaccination. *PLoS computational biology*, 5(2):e1000280, 2009.

[31] Kalyan S Perumalla and Sudip K Seal. Discrete event modeling and massively parallel execution of epidemic outbreak phenomena. *Simulation*, page 0037549711413001, 2011.

[32] Snyder Peter. tmpfs: A virtual memory file system. pages 241–248, 1990.

[33] P.J.Haas. Improving efficiency of stochastic composite simulation models via result caching. 2014. Submitted for publication.

[34] R.Talmage. Partitioned table and index strategies using sql server 2008, 2014.

[35] Thomas C Schelling. Dynamic models of segregation. *Journal of mathematical sociology*, 1(2):143–186, 1971.

[36] Vinoth Suryanarayanan and Georgios Theodoropoulos. Synchronised range queries in distributed simulations of multiagent systems. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 23(4):25, 2013.

[37] Inc Sybase. Performance and tuning: Optimizer and abstract plans, 2003.

[38] A SZALA. Science in an exponential world. *Nature*, 440:2020, 2006.

[39] Sebastiaan J van Schaik, Dan Olteanu, and Robert Fink. Enframe: A platform for processing probabilistic data. *arXiv preprint arXiv:1309.0373*, 2013.

[40] Guozhang Wang, Marcos Vaz Salles, Benjamin Sowell, Xun Wang, Tuan Cao, Alan Demers, Johannes Gehrke, and Walker White. Behavioral simulations in mapreduce. pages 952–963. ACM, VLDB Endowment, 2010.

[41] Walker White, Alan Demers, Christoph Koch, Johannes Gehrke, and Rajmohan Rajagopalan. Scaling games to epic proportions. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 31–42. ACM, 2007.

[42] WHO. Ebola virus disease, west africa update, 2014.

[43] Bing Wu, Kaishu Tai, Stuart Murdock, Muan Hong Ng, Steve Johnston, Hans Fangohr, Paul Jeffreys, Simon Cox, Jonathan Essex, and Mark SP Sansom. Biosimgrid: a distributed database for biomolecular simulations. 2003.

[44] 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 *Proceedings of the IEEE International Parallel & Distributed Processing Symposium (to appear)*, 2014.

[45] Bernhard Zeller and Alfons Kemper. Experience report: exploiting advanced database optimization features for large-scale sap r/3 installations. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 894–905. VLDB Endowment, 2002.