AUTONOMIC WEB-BASED SIMULATION

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Yingping Huang, M.S.

Gregory Madey, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana
March 2005

AUTONOMIC WEB-BASED SIMULATION

Abstract

by

Yingping Huang

Many scientific and engineering simulations are large programs which despite careful debugging and testing will probably contain errors when deployed to the Web for use. Based on the assumption that such scientific and engineering simulations do contain errors and that the underlying computing systems do fail due to hardware or software errors, we present a framework called autonomic web-based simulation (AWS), a supporting data warehouse, and their implementations to develop and deploy reliable web-based simulations. AWS strives to achieve the following features presented in the Vision of Autonomic Computing: self-configuring, self-optimizing, self-healing and self-protecting.

We discuss mathematical models to simulate the execution of scientific simulations and formulate objective functions for the purpose of determining optimal checkpoint intervals. Checkpointing is a basic requirement for self-healing of AWS and determining optimal checkpoint interval helps achieve self-optimizing of AWS.

A novel three step data cleansing algorithm is designed through approximate string joins. A self-manageable system is implemented for the NOM simulation project which allows scientists and other end users run NOM simulations anytime from anywhere.

CONTENTS

TABLES

FIGURES

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Madey for his academic guidance and support. I appreciate his patience and confidence in me.

I would also like to thank my collaborators Dr. S. Cabaniss, L. Arthurs, X. Xiang, R. Kennedy and Dr. P. Maurice. They are supportive, helpful and a great source of ideas.

I especially want to thank my family, Qiuju and Maggie, because without their support I would have unlikely finished this dissertation.

I want to thank my department chair, Dr. Alexander, University of North Alabama, for her support and encouragement to finish my dissertation.

Finally a big thank goes to all my committee members for their patience in reading this dissertation.

This research was partially supported by the NFS ITR Grant 0112820 and the Center for Environmental Science and Technology at University of Notre Dame.

CHAPTER 1

INTRODUCTION

Scientific and engineering simulation is the process of designing a model of a real system and conducting experiments on this model for the purpose of understanding the behavior of the system or of evaluating various strategies for the design and operations of the system. Scientific simulations have been increasingly applied to solve a variety of scientific problems. Domains such as biology, bioinformatics, chemistry, and environmental science are benefiting from this capability.

Most simulations models currently available run in stand-alone or traditional client-server architecture. Both of these models require installing software on the user's computers. This presents a significant barrier due the incompatibility that complicates or prevents installation. Moreover, the stand-alone or traditional client-server approach have significant drawbacks: (1) lack of collaborations and information sharing among users, (2) lack of reliability since usually no fault-tolerant features are built into simulations, and (3) lack of centralized simulation management and data analysis.

To overcome these drawbacks, Web-based simulations have been developed and deployed recently using server-side technologies [60]. However, developers of large-scale web-based scientific simulations have experienced increased complexity in their software systems due to the complex integration of different pieces of services. Web-based simulations have to be deployed to the Web through computing systems.

Such computing systems often consist of compiled simulation programs, storage devices, network, databases, middleware and other servers. These components of the computing systems often have workflow dependencies and interactions among themselves. Managing such a computing system involves configuring the individual components so that the overall system goals can be achieved. In his 2003 Turing Award speech, "What next? - a dozen information technology research goals", J. Gray, from Microsoft Research, emphasized the need for self-manageable systems in which the administrator sets system goals and creates high-level policies, while the system by itself decides how they can be achieved [48].

In 2001, IBM launched the **Autonomic Computing** initiative to handle this increased complexity [72]. Autonomic Computing is a vision striving for system self-management, which has the following four features: self-configuring, self-healing, self-protecting and self-optimizing.

Actually, all of these features have been under constant investigation by researchers for a long time [34, 106, 123, 16, 86, 125, 26, 107, 28, 120, 4]. Autonomic Computing tries to unify them into system self-management. The benefits of autonomic computing include stability, high availability and security, and fewer system or network errors due to self-healing.

Web-based simulations could benefit from system self-management. In this dissertation, we design a robust framework called autonomic web-based simulations (AWS) to develop and deploy Web-based scientific simulations based on the vision of Autonomic Computing. The power of simulation is the ability to model the dynamics of a real system and to analyze the results. It is important to analyze the simulation data so that the output of the simulation is not misinterpreted. The complexity of simulation data often requires more sophisticated analysis other than statistical analysis, such as data warehousing and data mining. With this in mind,

we incorporate the techniques of data cleansing, data warehousing and data mining into the framework for better analysis of simulation data.

The focus of this dissertation is to investigate and design robust methods to build reliable web-based simulations with the presence of hardware or software failure. This dissertation does not strive to solve the entire problem of autonomic computing, because it is believed to be the next era of computing, but instead focuses on select autonomic computing capabilities for scientific and engineering simulations. In its 2001 Autonomic Computing Manifesto [62], IBM calls on the entire IT industry to refocus its priorities on cooperating in developing the necessary standards and open interfaces to make the Autonomic Computing vision a reality.

1.1 Dissertation Contributions

The work described in this dissertation was initiated by a research project, "Stochastic Synthesis: Simulating the Environmental Transformations of Natural Organic Matter", which is supported in part by the National Science Foundation, Information Technology Research (ITR/AP-DEB). This project is a multi-disciplinary project involving chemists, biologists, geologists, environmental scientists, and computer scientists.

The dissertation's goals are to design and implement a self-managing system to support web-based simulations so that scientists and other end users can run scientific simulations and obtain simulation reports anytime from anywhere. Reliability and efficiency are major concerns in the design of this system.

The main contributions of this dissertation are

- **Presentation of three mathematical models to simulate the executions of scientific simulations**: Objective functions on total expected execution time are formulated. Then the optimal checkpoint intervals are calculated by minimizing the objective function. Experiments show that the third model

is effective to simulate the lifecycle of executions of simulations. Through this model we found that the optimal checkpoint interval is independent of the simulation restart time, while the total expected execution time is exponentially dependent on the restart time.

- **A framework called autonomic web-based simulations, and a supporting data warehouse, and their implementations**: Three basic requirements to achieve autonomic web-based simulations are proposed. These requirements include 1) simulation checkpoint and restart, 2) proactive failure detection using Java management and monitoring APIs, and 3) a self-managing computing infrastructure for hosting web-based simulations.

- **A novel three-step data cleansing algorithm is designed through approximate string joins**: The strings are first mapped into high dimensional Euclidean space, then an approximate matrix multiplication method is used to find close pairs of points in the Euclidean spaces. Finally, the distances between preimages of these pairs of points are evaluated to identify the close pairs of strings. Experiments are conducted on real data sets, which show that the approach is both effective and efficient compared to other approaches.

- **A complete system for NOM (natural organic matter) simulations is implemented using the idea of autonomic web-based simulations**: A portal is designed using state-of-the-art technologies so that scientists and other end users can run simulations and obtain reports any time from anywhere. A data warehouse is also designed for scientific simulation data analysis.

1.2 Dissertation Outline

The rest of this dissertation is organized as follows:

Chapter 2 presents three mathematical models for simulation checkpointing. A best model is selected to determine the simulation checkpoint intervals optimally. Checkpointing is a basic requirement for self-healing, while optimal checkpoint in-

terval selection helps achieve self-optimizing. Both self-healing and self-optimizing are features of autonomic web-based simulations.

Chapter 3 proposes a framework called autonomic web-based simulation (AWS) for developing and deploying reliable web-based simulations. It first presents three requirements for realizing autonomic web-based simulations. It then shows a prototype implementation of AWS so that self-configuring, self-healing, self-optimizing and self-protecting are achieved.

Chapter 4 describes the design of a data warehousing to support autonomic web-based simulation. The data warehouse provides necessary tools to analyze information for the underlying computing infrastructure so as to achieve system self-management.

Chapter 5 proposes a general data cleansing algorithm using approximate string joins, and its implementation using pure SQL. Experiments are conducted on real world data sets to validate the efficiency of this data cleansing algorithm.

Chapter 6 uses the NOM simulation [13] as a case study to evaluate the effectiveness of autonomic web-based simulation. We also describe our experience in designing a scientific simulation data warehouse to better analyze and use simulation data.

Chapter 7 concludes this dissertation. A summary of contributions is presented.

CHAPTER 2

CHECKPOINTING FOR AUTONOMIC WEB-BASED SIMULATIONS

Checkpointing helps achieve self-healing and self-optimizing of autonomic web-based simulations. Long-running scientific simulations may crash before completion. It's very costly to restart them from scratch in case of failure. Checkpointing provides a solution so that simulation can be restarted from near the point of failure. However, it is not trivial to determine the optimal interval between contiguous checkpoints. In this chapter, we discuss three models to calculate the optimal checkpoint interval and predict the expected execution time of scientific simulations.[1]

2.1 Introduction

Many scientific simulations are long-running and may run for hours, days, or even months. It's very costly to restart a simulation from scratch if it dies prematurely. To prevent restarting from the beginning, a mechanism called **checkpointing** is used to save the state of the simulation periodically. Checkpoint and restart strategies have been under continuous investigation in the simulation, systems, and database communities. The papers by Chandy [17] and Nicola [89] give excellent overviews of checkpointing and recovery strategies in the literature.

Scientific simulations would benefit from this simple checkpointing mechanism

[1]Part of this chapter appeared in ANSS38 [58], and a full version is under review with International Journal of Modeling and Simulation

that provides automatic restart or recovery in response to faults and failures, and enables dynamic load balancing and improved resource utilization through simulation migration [60, 75]. However it is usually not a trivial task to optimally determine the "interval" between contiguous checkpoints. Excessive checkpointing would result in performance degradation and thus longer completion time, while deficient checkpointing would incur an expensive recovery overhead and thus again longer completion time. Therefore, a trade-off must be made to determine the checkpoint interval optimally. We define **checkpoint interval** to be the time between two consecutive checkpoints. In this chapter, we present models to analytically determine the optimal checkpoint interval.

The rest of the chapter is organized as follows: Section 2.2 discusses the execution of a simulation; Sections 2.3 to 2.5 propose three models to model the executions of scientific simulations; Section 2.6 presents the experiments and discussions on these models; Section 2.7 reviews some of the checkpoint and recovery strategies in the literature; and finally, Section 2.8 draws conclusions.

2.2 The Execution of a Simulation

Figure 2.1 shows the execution of a simulation, where each *xc* or *rxc* is called an **execution segment**. We call it an *xc*-segment or *rxc*-segment respectively. The execution lifecycle of a simulation may include a sequence of checkpoints and possible restarts. Failures may occur any time during the execution of simulations. Once a failure occurs, the failure is detected and the simulation is restarted from the most recent checkpoint. From our experience, users(scientists) may even request to continue a previously completed simulation so that more simulation data can be produced and studied. As shown in Figure 2.1, when a simulation completes, its final state is checkpointed.

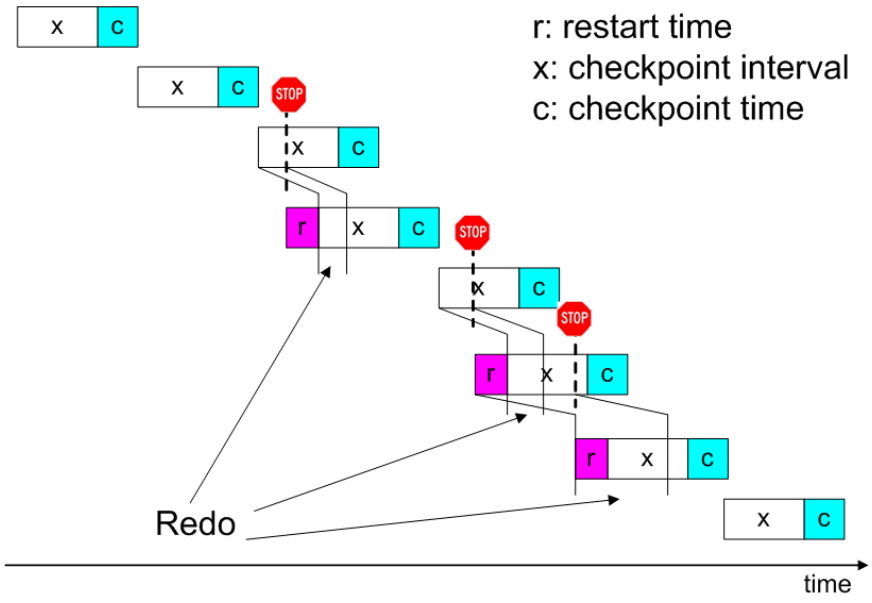

Figure 2.1. Execution of scientific simulations: r is the restart time, c is the checkpoint time and x is the checkpoint interval

The total execution time of a simulation can be partitioned into the following four parts:

- Work Time (denoted by T_{work}): Time needed to complete a simulation based on the assumption that the simulation never crashes and thus no checkpoint is necessary;
- Checkpoint Time (denoted by $T_{checkpoint}$): Time spent to write checkpoint data to files or databases;
- Redo Time (denoted by T_{redo}): Time spent to redo the simulation from the most recent checkpoint to the point of failure;
- Restart Time (denoted by $T_{restart}$): Time needed to detect the failures and restore simulation states from checkpoint data (either data files or databases) so that redo can proceed.

The total execution time (denoted by T_{total}) is thus

$$T_{total} = T_{work} + T_{checkpoint} + T_{redo} + T_{restart} \tag{2.1}$$

For scientific simulations, T_{work} is either explicitly specified by the user as an input or can be derived implicitly based on some terminating conditions. We denote T_{work} by N, and the checkpoint interval to be determined by x. To analytically derive the optimal checkpoint interval x^*, we make the following assumptions:

- The average time before a crash occurs is M, and crashes occur according to a Poisson process with rate $\frac{1}{M}$. More precisely, (1) crashes occur randomly, but with a long-term average of one crash per M time units; (2) the likelihood of a crash is independent of the past history; (3) crashes are rare in a very short time interval, and there is a negligible chance of more than one crash in a very short time interval. This assumption is widely used in the literature related to checkpointing strategies, such as [77, 39, 116]. Based on these assumptions, the probability that a simulation successfully completes t time units is $p(t) = e^{-\frac{t}{M}}$.
- Assume the checkpoint time is c and the restart time is r, where c and r are constants for all xc-segments or rxc-segments.

Let n be the expected number of failures occurring during the execution of a simulation, and let f be the fraction of redo time over the time of an execution segment when a crash occurs. At this point, we also assume that crash does not occur during an rxc-segment, i.e., crash does not occur immediately after a restart; however this assumption will be removed in Section 2.5 when we present the best model. Based on the above assumptions, we have the following facts:

- $T_{work} = N$
- $T_{checkpoint} = \frac{Nc}{x}$
- $T_{redo} = f \cdot (x + c) \cdot n$
- $T_{restart} = r \cdot n$

Note that the number of execution segments without failures is $\frac{N}{x}$.

Thus the expected total execution time is

$$T_{total} = N + \frac{Nc}{x} + f \cdot (x + c) \cdot n + r \cdot n \tag{2.2}$$

We need to derive n and f, so that we can analytically determine x to minimize T_{total}. We obtain the following:

- The probability to successfully complete an xc-segment without crash is $p(x + c) = e^{-\frac{x+c}{M}}$.
- Therefore, the expected number of execution segments to complete N time units is $\frac{N}{x \cdot p(x+c)} = \frac{N}{x} e^{\frac{x+c}{M}}$.
- Thus, the expected number of failures n is $n = \frac{N}{x \cdot p(x+c)} - \frac{N}{x} = \frac{N}{x}\left(e^{\frac{x+c}{M}} - 1\right)$.
- Suppose z is the time of an execution segment (could be either $x+c$ or $r+x+c$). The distribution of failures occurring at t after the most recent checkpoint is

$$\begin{aligned} d(t) &= \sum_{i=0}^{\infty} \frac{1}{M} e^{-\frac{t+i \cdot z}{M}} \\ &= \frac{e^{-\frac{t}{M}}}{M\left(1 - e^{-\frac{z}{M}}\right)} \end{aligned}$$

- Therefore, the expected point of failure between 0 and z is

$$\begin{aligned} E(z) &= \int_0^z t \cdot d(t) dt \\ &= \frac{\int_0^z t \cdot e^{-\frac{t}{M}} dt}{M\left(1 - e^{-\frac{z}{M}}\right)} \\ &= M + \frac{z}{1 - e^{\frac{z}{M}}} \end{aligned}$$

- Thus, the expected fraction of redo over z time units is

$$f(z) = \frac{M}{z} + \frac{1}{1 - e^{\frac{z}{M}}} \tag{2.3}$$

Before we substitute n and f into equation 2.2, let's examine some properties of f. Let $y = \frac{M}{z}$, and let $g(y) = y + \frac{1}{1 - e^{\frac{1}{y}}}$. We have the following lemma:

Lemma 2.1. *$g(y) = y + \frac{1}{1-e^{\frac{1}{y}}}$, $y > 0$, is monotonely increasing and $\lim_{y\to\infty} g(y) = \frac{1}{2}$.*

Proof. To prove $g(y)$ is monotonely increasing, it suffices to prove that $\frac{dg(y)}{dy} > 0$. In fact,

$$\begin{aligned}\frac{dg(y)}{dy} > 0 &\iff 1 - \frac{\frac{1}{y^2}e^{\frac{1}{y}}}{\left(1-e^{\frac{1}{y}}\right)^2} > 0 \\ &\iff y\left(e^{\frac{1}{y}} - 1\right) > e^{\frac{1}{2y}}\end{aligned}$$

Expand both sides of the last inequality using Taylor series, then

$$y\left(e^{\frac{1}{y}} - 1\right) = 1 + \frac{1}{2y} + \sum_{i=2}^{\infty} \frac{1}{(i+1)!y^i}$$

and

$$e^{\frac{1}{2y}} = 1 + \frac{1}{2y} + \sum_{i=2}^{\infty} \frac{1}{2^i i!y^i}$$

It's easy to see that $2^i > (i+1)$ when $i \geq 2$, and thus $(i+1)!y^i < 2^i i!y^i$. Therefore, $y\left(e^{\frac{1}{y}} - 1\right) > e^{\frac{1}{2y}}$, and hence $g(y)$ is monotonely increasing. And

$$\begin{aligned}\lim_{y\to\infty} g(y) &= \lim_{y\to\infty} \frac{\left(e^{\frac{1}{y}} - 1\right)y - 1}{e^{\frac{1}{y}} - 1} \\ &= \lim_{y\to\infty} \frac{\left(\sum_{i=0}^{\infty} \frac{1}{i!y^i} - 1\right)y - 1}{\sum_{i=0}^{\infty} \frac{1}{i!y^i} - 1} \\ &= \lim_{y\to\infty} \frac{\sum_{i=1}^{\infty} \frac{1}{(i+1)!y^i}}{\sum_{i=1}^{\infty} \frac{1}{i!y^i}} \\ &= \lim_{y\to\infty} \frac{\frac{1}{2} + \sum_{i=1}^{\infty} \frac{1}{(i+2)!y^i}}{1 + \sum_{i=1}^{\infty} \frac{1}{(i+1)!y^i}} = \frac{1}{2}\end{aligned}$$

□

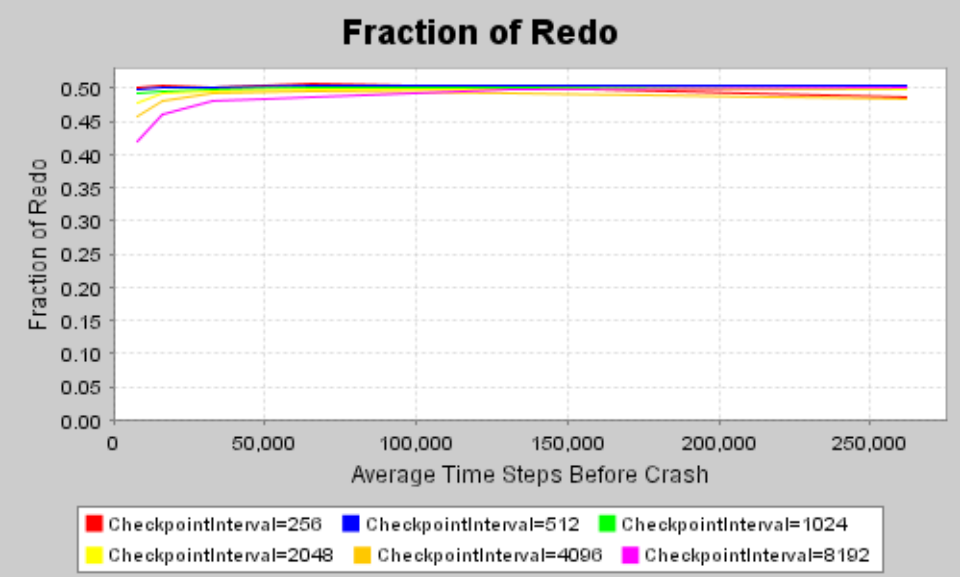

Figure 2.2. Average fraction of redo over an execution segment of a simulation

Figure 2.2 confirms that for fixed checkpoint interval x, the average fraction of redo over an execution segment converges to $\frac{1}{2}$ as M increases to positive infinite. Figure 2.2 is generated using JFreeChart through experiments on running simulations. Next we derive our models to determine the optimal checkpoint intervals for autonomic web-based simulations.

2.3 Model I

We assume that M is sufficiently large compared to x, c, r. By Lemma 2.1, we see that $f(x+c)$ is approximately $\frac{1}{2}$. Substitute $f = \frac{1}{2}$ and $n = \frac{N}{x}\left(e^{\frac{x+c}{M}} - 1\right)$ into equation 2.2, we obtain

$$T_{total}(x) = N + \frac{Nc}{x} + \left(\frac{1}{2} \cdot (x+c) + r\right) \cdot \frac{N}{x}\left(e^{\frac{x+c}{M}} - 1\right) \tag{2.4}$$

We need to find x^* so that $T_{total}(x)$ is minimized, which suffices to calculate x such that $\frac{dT_{total}(x)}{dx} = 0$. We then have

$$e^{\frac{x+c}{M}}(x^2+(c+2r)x-(c+2r)M)+(2r-c)M=0. \tag{2.5}$$

It's unlikely to find an exact solution for this equation analytically. To asymptotically solve it, we need the following lemma:

Lemma 2.2. $-\log y \approx 1-y$ *if* $0<y<1$ *and* $y\approx 1$.

Proof. Use $\lim_{x\to 0}\frac{\log(1-x)}{x}=-1$. □

By Lemma 2.2, equation 2.5 can be written as

$$\begin{aligned}\frac{x+c}{M} &= -\log\frac{x^2+(c+2r)x-(c+2r)M}{(c-2r)M}\\ &\approx 1-\frac{x^2+(c+2r)x-(c+2r)M}{(c-2r)M}.\end{aligned}$$

With standard algebraic calculations, the above equation can be simplified to

$$(x+c)^2=2(M+r)c$$

Thus,

$$x=\sqrt{2(M+r)c}-c \tag{2.6}$$

Hence we have the following:

Theorem 2.1. *The optimal checkpoint interval so that the total execution time is minimized is* $x^*=\sqrt{2(M+r)c}-c$. *And the expected total execution time is* $T_{total}(x^*)$ *where* T *is in Equation 2.4.*

2.4 Model II

Instead of approximating f with $\frac{1}{2}$, we now substitute $f=f(x+c)=\frac{M}{x+c}+\frac{1}{1-e^{\frac{x+c}{M}}}$ and $n=\frac{N}{x}\left(e^{\frac{x+c}{M}}-1\right)$ into equation 2.2, we obtain

$$\begin{aligned}T_{total}(x) &= N+\frac{Nc}{x}\\ &+\left(\left(\frac{M}{x+c}+\frac{1}{1-e^{\frac{x+c}{M}}}\right)(x+c)+r\right)\frac{N}{x}\left(e^{\frac{x+c}{M}}-1\right)\end{aligned}$$

After standard algebraic transformations, the above equation can be simplified as

$$T_{total}(x) = \frac{N(M+r)}{x}\left(e^{\frac{x+c}{M}} - 1\right) \tag{2.7}$$

Again, to minimize $T_{total}(x)$, we take the first derivative of equation 2.7 and let it be zero, or equivalently

$$\frac{x+c}{M} = -\log\left(1 - \frac{x}{M}\right) \tag{2.8}$$

If x^* is the solution for the above equation, then $\frac{d^2T_{total}(x^*)}{dx^2} = \frac{N(M+r)}{Mx^{*2}(M-x^*)} > 0$. Hence x^* achieves the minimum for $T_{total}(x)$[2]. It's far from trivial to solve Equation 2.8 analytically. However, it is extremely simple to solve it numerically based on the fact in the following lemma:

Lemma 2.3. *Equation 2.8 has one and only one solution in the interval* $(0, M)$.

Proof. Let $g(x) = \frac{x+c}{M} + \log\left(1 - \frac{x}{M}\right)$. Then $g(0+) = \frac{c}{M} > 0$, and $g(M-) = \frac{M+c}{M} + \log(0+) = -\infty < 0$. Since $g(x)$ is continuous, there exists x in the interval $(0, M)$ such that $g(x) = 0$. Furthermore, $\frac{dg(x)}{x} = \frac{1}{M}\left(1 - \frac{1}{1-\frac{x}{M}}\right) < 0$ for any $x \in (0, M)$, which means that $g(x)$ is monotonely decreasing in the interval $(0, M)$. Hence, there exists one and only one x such that $g(x) = 0$. Thus, the lemma holds. □

From the proof of the above lemma, we see that given values of M and c, we can numerically solve the equation $g(x) = 0$ using a simple bisection algorithm. In the following algorithm, ϵ is typically chosen as 0.0001.

1: set $t_{lo} = 0$ and $t_{hi} = M$.
2: **while** $(t_{hi} - t_{lo} > \epsilon)$ **do**
3: $\quad t_{mi} = \frac{1}{2}(t_{lo} + t_{hi})$
4: $\quad$ **if** $g(t_{lo})g(t_{mi}) > 0$ **then**
5: $\quad\quad t_{lo} = t_{mi}$
6: $\quad$ **else**

[2]If the first derivative $f'(x) = 0$ and the second derivative $f''(x) > 0$, then x is a local minimum.

7: $t_{hi} = t_{mi}$

8: **end if**

9: **end while**

Even if we cannot solve the equation 2.8 analytically, we can asymptotically solve it. To find an asymptotic solution, we consider two cases:

- Case I: $\frac{c}{M} \to 0+$, or $c \ll M$, i.e., the checkpoint time c is much less than the average time before crash M. Then $\frac{x}{M} \to 0+$ according to equation 2.8. Thus
$$\frac{x+c}{M} = -\log\left(1 - \frac{x}{M}\right) \approx \frac{x}{M} + \frac{x^2}{2M^2}$$
Thus, we obtain
$$x = \sqrt{2Mc} \tag{2.9}$$

- Case II: Now we consider the general case. Let $d = \sqrt{\frac{2c}{M}}$ and $y = \frac{x}{M}$. Write y as
$$y = \sum_{n=0}^{\infty} a_n d^n \tag{2.10}$$
From Case I, we see that $y \to d$ as $d \to 0$. Thus we have $a_0 = 0$ and $a_1 = 1$. Expand $\log(1 - \frac{x}{M})$ using Taylor series. Equation 2.8 becomes
$$\frac{1}{2}d^2 - \sum_{n=2}^{\infty} \frac{y^n}{n} = 0 \tag{2.11}$$
To obtain a asymptotic solution, we now let $y = d + a_2 d^2 + a_3 d^3 + o(d^3)$ and expand equation 2.11 [3] . Equating the terms of powers of d, we have $a_2 = -\frac{1}{3}$, and $a_3 = \frac{1}{36}$. Thus $y = d - \frac{d^2}{3} + \frac{d^3}{36} + o(d^3)$. Therefore
$$x \approx \sqrt{2Mc}\left(1 - \frac{1}{3}\sqrt{\frac{2c}{M}} + \frac{c}{18M}\right) \tag{2.12}$$

Hence we have the following

Theorem 2.2. *The optimal checkpoint interval x^* that minimizes the total execution time is asymptotically $\sqrt{2Mc}\left(1 - \frac{1}{3}\sqrt{\frac{2c}{M}} + \frac{c}{18M}\right)$. And the expected total execution time is $N \cdot \frac{1+\frac{r}{M}}{1-\frac{x^*}{M}}$. It's interesting to note that the optimal checkpoint interval is independent of the restart time r.*

[3] We are certainly able to approximate the solution to higher orders. However, this suffices to demonstrate the idea of obtaining asymptotic solutions

2.5 Model III

In Model I and Model II, we assumed that failure does not occur during *rxc*-segments. In this section, we remove this assumption so that a crash may occur in both *xc*-segments and *rxc*-segments.

The probability that an *rxc*-segment completes without failure is $e^{-\frac{r+x+c}{M}}$. Thus the probability that a failure does occur in an *rxc*-segment is $1 - e^{-\frac{r+x+c}{M}}$. For a simulation with total execution time units T_{total} and average time before crash M, the expected number of failures is $\frac{T_{total}}{M}$.

Therefore, the expected number of failures to occur in *rxc*-segments is $n_{rxc} = \frac{T_{total}}{M}\left(1 - e^{-\frac{r+x+c}{M}}\right)$, and the expected number of failures to occur in *xc*-segments is $n_{xc} = \frac{T_{total}}{M}e^{-\frac{r+x+c}{M}}$. Note that a restart time r is added after a failure in an *xc*-segment; while a restart time r is not added after a failure in an *rxc*-segment since it is already included. Therefore the total execution time is now

$$\begin{aligned} T_{total}(x) &= N + \frac{Nc}{x} + (f(x+c)(x+c) + r)n_{xc} \\ &+ (f(r+x+c)(r+x+c))n_{rxc} \end{aligned}$$

Substitute f, n_{xc} and n_{rxc} into the above equation and simplify it with a series of algebraic transformations[4], we obtain

$$T_{total}(x) = MNe^{\frac{r}{M}}\frac{e^{\frac{x+c}{M}} - 1}{x} \tag{2.13}$$

As before, we take the first derivative and let it be zero, then we obtain

$$\frac{x+c}{M} = -\log\left(1 - \frac{x}{M}\right)$$

Surprisingly, we have the same minima for Model III and Model II. Hence we have the following

[4]See Appendix F for the derivation

Theorem 2.3. *The optimal checkpoint interval x^* that minimizes the total execution time is asymptotically $\sqrt{2Mc}\left(1-\frac{1}{3}\sqrt{\frac{2c}{M}}+\frac{c}{18M}\right)$. And the expected total execution time is $N\cdot\frac{e^{\frac{r}{M}}}{1-\frac{x^*}{M}}$. It's interesting to note that the optimal checkpoint interval is independent of the restart time r.*

We see that T_{total} in Model II and Model III have the same minima. Let T_2=$T_{total}(x^*)$ in Model II and $T_3 = T_{total}(x^*)$ in Model III. Then

$$\frac{T_3}{T_2} = \frac{e^{\frac{r}{M}}}{1+\frac{r}{M}} = \frac{1+\frac{r}{M}+\sum_{n=2}^{\infty}\frac{r^n}{n!M^n}}{1+\frac{r}{M}} > 1$$

This means that although Model II and Model III have the same argmins, the expected total execution time is longer for Model III than Model II. Note that $\lim_{x\to 0}\frac{e^x}{1+x}=1$, thus $T_2 \approx T_3$ if the restart time r is far less than M. In other words, if $r \ll M$, then there is no distinguishable difference between Model II and III.

2.6 Experiments and Discussion

A simple Java simulation is developed to experiment with and evaluate the above three models. The simulation first generates random points of failure according to a Poisson process with rate $\frac{1}{M}$, then outputs the total execution time as a function of N, M, r, x, and c. We run the simulation 1000 times for each distinct combination of N, M, r, x, and c. Appendix A shows the Java implementation of this simulation. The complete code that generates all experiment data can be downloaded from http://www.nd.edu/~nom/ckpt.zip. Now we present the following experiments:

- Experiment 1: We set $N = 100000$, $M = 2000$, $r = 20$ and $c = 10$. The experimental results are shown in Figure 2.3. Since r and c are small compared to M, we see that all three models are good matches of the simulation results. In other words, there is little difference between the three models if $\frac{r}{M}$ and $\frac{c}{M}$ are small. The predicted optimal checkpoint interval for Model I is 191. And the predicted optimal checkpoint intervals for Model II and III are 193.

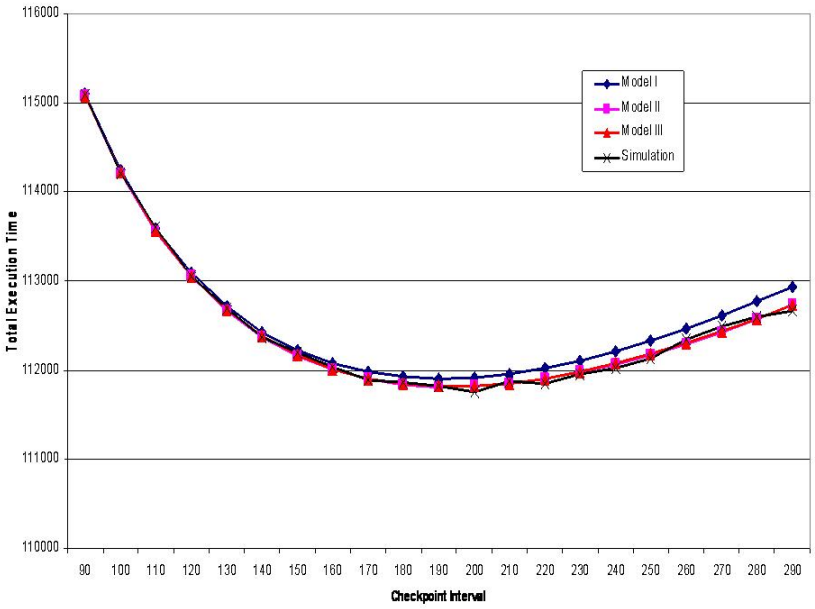

Figure 2.3. Experiment 1: $N = 100000$, $M = 2000$, $r = 20$, and $c = 10$

- Experiment 2: Now we increase the checkpoint time c in Experiment 1, and set $N = 100000$, $M = 2000$, $r = 20$ and $c = 100$. The experimental results are shown in Figure 2.4. We see that Model I deviates from the simulation results. However, both Model II and Model III are still in good agreement with the simulation results. Since $r \ll M$, there is no distinguishable difference between Model II and III. The predicted checkpoint interval for Model I is 536. And the predicted optimal checkpoint intervals for Model II and III are 568.

- Experiment 3: We increase the restart time r in Experiment 2 and set $N = 100000$, $M = 2000$, $r = 200$ and $c = 100$. The experimental results are shown in Figure 2.5. We see that both Model I and Model II deviate from the simulation results. However, Model III is still in good agreement with the simulation results. The predicted checkpoint interval for Model I is 563. And the predicted checkpoint intervals for Model II and III are 568.

From the above three experiments, we see that Model III is the best one to match the simulation results among all the three models. When the restart time r

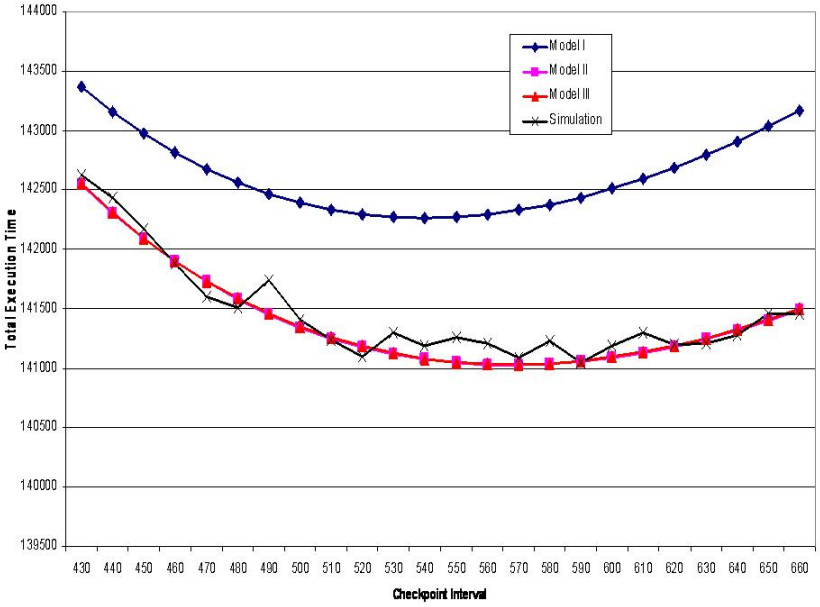

Figure 2.4. Experiment 2: $N = 100000$, $M = 2000$, $r = 20$, and $c = 100$

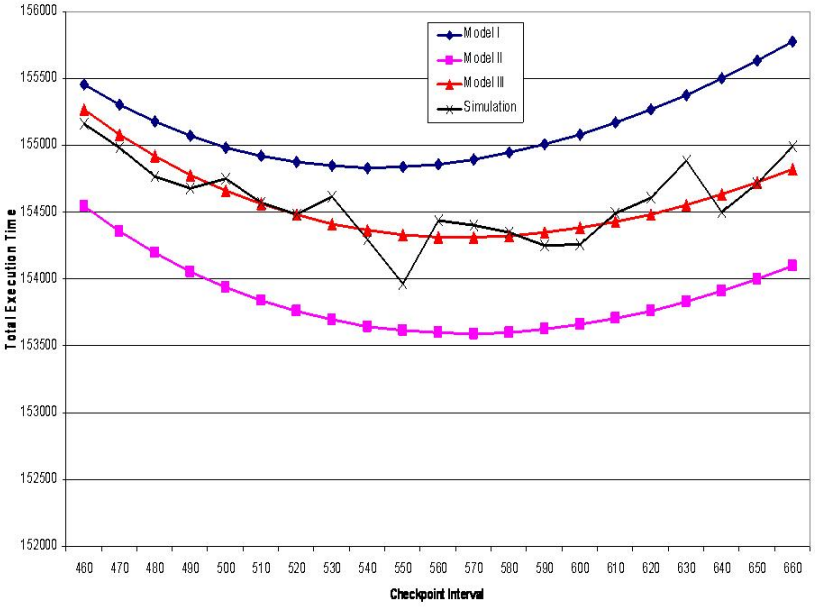

Figure 2.5. Experiment 3: $N = 100000$, $M = 2000$, $r = 200$, and $c = 100$

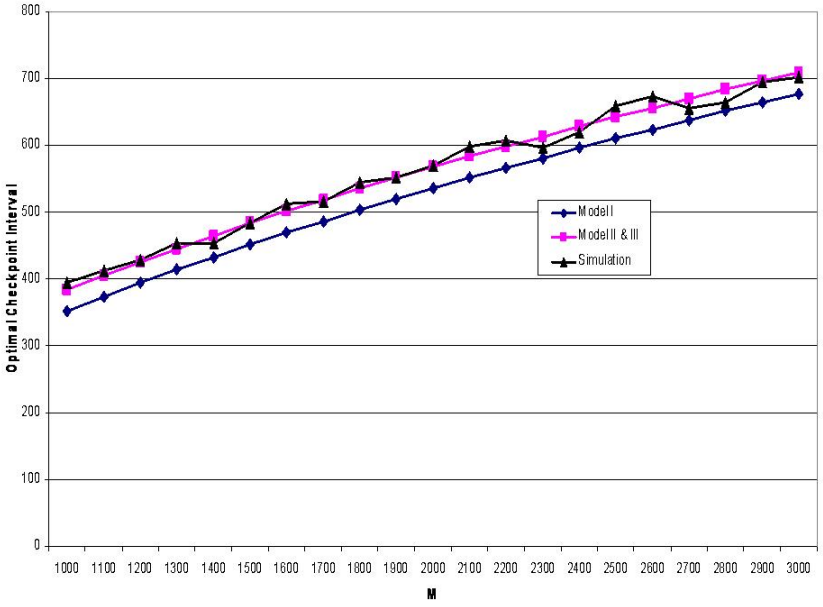

Figure 2.6. Experiment 4: $N = 100000$, $r = 20$, and $c = 100$

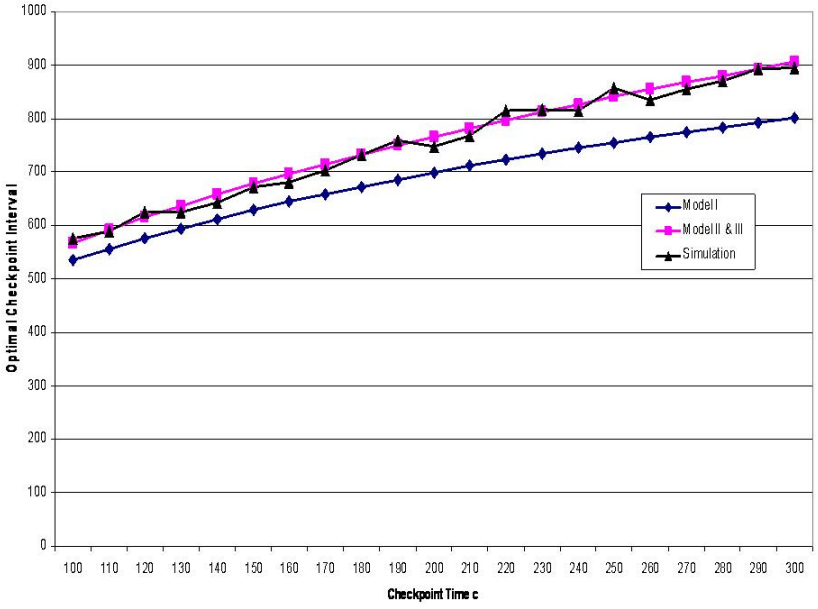

Figure 2.7. Experiment 5: $N = 100000$, $M = 2000$, and $r = 20$

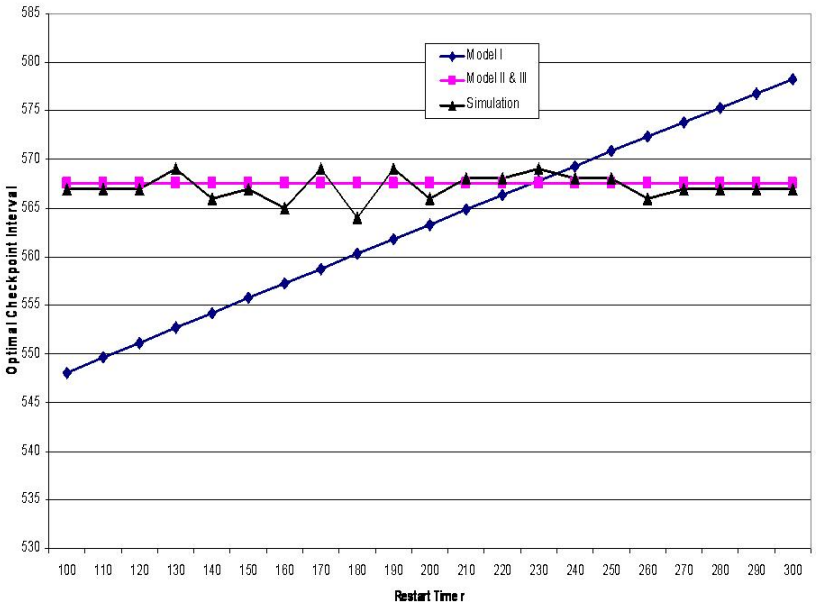

Figure 2.8. Experiment 6: $N = 100000$, $M = 2000$, and $c = 100$

and checkpoint c are $\ll M$, the three models are all in good agreement with the simulation results.

Next we experiment with how the changes of average time before crash M, checkpoint time c and restart time r affect the choice of the optimal checkpoint interval. The optimal checkpoint interval for the simulation is obtained in the following way: first we calculate the optimal checkpoint interval $xopt$ from Model II (or Model III, since they are the same); then we run the simulation 1000 times for each checkpoint interval in a wide neighborhood (for example from $xopt - 100$ to $xopt + 100$) of $xopt$, and the checkpoint interval that results in least total execution time is believed to be the optimal checkpoint interval for the simulation.

- Experiment 4: We set $N = 100000$, $r = 20$ and $c = 100$, and let M range from 1000 to 3000. Figure 2.6 shows the experimental results of the relationship between M and the optimal checkpoint interval x^*. From the figure, we

see that x^* monotonely increases as M increases for all three models. And the optimal checkpoint interval calculated from Model II (or III) is in good agreement with that from the simulation.

- Experiment 5: We set $N = 100000$, $r = 20$ and $M = 2000$, and let c range from 100 to 300. Figure 2.7 shows the experimental results of the relationship between c and the optimal checkpoint interval x^*. From the figure, we see that x^* monotonely increases as c increases for all three models. And the optimal checkpoint interval calculated from Model II (or III) is in good agreement with that from the simulation.

- Experiment 6: We set $N = 100000$, $M = 2000$ and $c = 100$, and let r range from 100 to 300. Figure 2.8 shows the experimental results of the relationship between r and the optimal checkpoint interval x^*. From the figure, we see that x^* monotonely increases as r increases for Model I, while stays constant for Model II and III. The simulation shows that x^* is almost constant, which confirms that the choice of optimal checkpoint interval is independent of the restart time r.

From these experiments, we conclude that Model III is a good model so that we can use it to calculate the optimal checkpoint interval and predict the total execution time of a simulation. M and c for a specific scientific simulation can be determined empirically by running the simulation a sufficient number of times. For example, in our case study of the NOM simulation, M is determined by running the simulation many times without checkpointing, and c is determined easily by calculating the average time spent on checkpointing.

One more interesting finding about Model III is the following lemma:

Lemma 2.4. *Let*

$$T(x) = NMe^{\frac{r}{M}} \frac{e^{\frac{x+c}{M}} - 1}{x} \tag{2.14}$$

where $c > 0$, $r > 0$ and $x \in (0, M)$. Suppose $x^* = \arg\min_{0<x<M} T(x)$, then for any $t > 0$ such that $0 < x^* - t < x^* + t < M$, we have $T(x^* - t) > T(x^* + t)$.

Proof. The proof is omitted since it's very tedious. See Appendix G for details. □

Lemma 2.4 means that an under-predicted checkpoint interval results in longer total execution time than an equivalent over-predicted one. This conclusion can be confirmed from Figure 2.3, Figure 2.4 and Figure 2.5.

2.7 Related Work On Checkpointing

In this section, we briefly review related work in the literature. Analytically determining optimal checkpoint dates back to as early as 1974[5] when Young presented a first order approximation to the optimum checkpoint interval. The first order approximation was $\sqrt{2cM}$ [100], which coincides with the special case in Equation 2.9. Our work extends this result to more general cases.

In several other papers [110, 31, 76], a model with Poisson failure is considered to determine the optimal number of checkpoints which minimizes the expected execution time of a program, with an assumption that no failure occurs during the checkpoint and restart phases. The same model is extended by [45] in which the optimal number of checkpoints relies on the distribution of the program execution time. Tantawi and Ruschitzka [116] consider a model with general distribution of failures and allow failures occurring during the checkpoint and restart phase. This generality yields a model that needs to compute an infinite number of integrals, which is computationally intractable. A simpler model is proposed by preventing failures from occurring during checkpoint and restart. However, this simplification still results in computing an infinite number of non-linear equations. Thus a even simpler model is then proposed by imposing more constraints which assume that the execution time between two successive checkpoints is constant and that the ex-

[5]The author cannot guarantee this was the earliest work on checkpointing, but a literature search on scholar.google.com returns this work as the earliest, and citations of this and other more recent papers do not return earlier citations

pected restart time equals the mean checkpointing time. With this simplification, an iteration algorithm with dynamic programming is used to compute an approximation of the optimal number of checkpoints. Ling et al [82] use a variational calculus approach to derive an explicit formula that links the optimal checkpointing frequency with a general failure rate, with the objective of globally minimizing the total expected cost of checkpointing and recovery. The results show that the optimal checkpointing frequency is proportional to the square root of the failure rate.

In time warp simulations, the state of each process must be checkpointed regularly in case a rollback is needed [80, 113, 102]. Lin and Lazowska [81] proposed a model to derive the optimal checkpoint interval by assuming that the rollback behavior of time warp is not affected by the frequency of checkpointing. An experimental study conducted by Preiss [96] indicates that this assumption is generally not valid. Lin et al [80] extend [81] to include the effect of the checkpoint interval on the rollback behavior. A checkpoint interval selection algorithm which determines the optimal checkpoint interval during the execution of time warp simulations is proposed. Palaniswamy [92] begins with a model similar to [80] and estimates the cost of a rollback due to periodic state saving. He models the average overhead during a given time interval. Minimizing the overhead function yields an expression for calculating an optimal checkpoint interval.

Our work differs from the aforementioned work in several ways: (1) our simple assumption (failures occur according to a Poisson process) yields a simple cost function which is easy to solve numerically using a simple bisection algorithm, and (2) experiments on simulation show that the model is in good agreement with simulations.

2.8 Summary

In this chapter, three models are discussed to determine the optimal checkpoint interval and predict total execution time for long-running scientific simulations. We can draw the following conclusions from the discussions:

- Model III is the best model that can be used to calculate the optimal checkpoint interval and predict total execution time.
- The choice of checkpoint interval is independent of the restart time r; however, the predicted total execution time is exponentially dependent on r. Therefore, the total execution time can be reduced dramatically if a failure can be detected and restart can be accomplished quickly.
- An under-predicted checkpoint interval results in longer total execution time than an equivalent over-predicted one; and therefore, we would rather choose a larger checkpoint interval if it's not possible or too difficult to calculate the optimal.

The ability to restart a failed simulation from the most recent checkpoint is a basic requirement for developing and deploying autonomic web-based simulations. In particular, it helps to achieve self-healing and self-optimizing features of autonomic web-based simulations. In the next chapter, we'll focus on a robust framework to build reliable web-based simulations, so that the simulations and underlying computing system can self-configure, self-heal, self-optimize and self-protect.

CHAPTER 3

AUTONOMIC WEB-BASED SIMULATIONS: A PROTOTYPE IMPLEMENTATION

Many scientific simulations are large programs which despite careful debugging and testing will probably contain errors when deployed to the Web for use. Based on the assumption that such scientific simulations do contain errors and the underlying computing systems do fail due to hardware or software errors, we investigate and design robust methods for building reliable systems to support web-based scientific simulations with the presence of such errors. In this chapter, we present a framework to build autonomic web-based simulation (AWS). Certain requirements must be satisfied during the development process of AWS. This chapter presents these requirements and shows how they can be satisfied. AWS strives to achieve the four features presented in the Vision of Autonomic Computing [72]: self-configuring, self-optimizing, self-healing and self-protecting.[1]

3.1 Background and Related Work

3.1.1 Examples of Web-based Simulations

With the emergence of the WWW, many web-based simulations and simulation platforms have been developed, such as [60, 121, 54]. In [60], we present a self-managing infrastructure to host scientific simulations. The infrastructure integrates

[1]Parts of this chapter appeared in ANSS37 [60], ANSS38 [58] and CITSA2004 [57], and a full version is under review with IEEE Computing in Science and Engineering

web servers, database servers, reports servers, data warehouses and data mining tools to provide a simulation and data analysis environment. The development of an integrated extensible web-based simulation environment called Computational Science and Engineering Online (CSEO) is presented in [121] to allow computational scientists to perform research on chemistry. In [54], the authors report that a metadata tools system and a data services system are undergoing development and integration at Sandia National Laboratories, to provide web-based access to high-performance computing clusters and its associated simulation data.

Some work focusing on simulation data analysis and understanding can be found in [1]. The authors describe scientific simulation data, its characteristic and the way scientists generate and use the data. Then they compare and contrast simulation data to data streams and claim that simulation data is a special case of data stream[2]. The authors present a tool called AQSim (Ad-hoc Queries for Simulation Data) system to analyze simulation data.

3.1.2 Autonomic Computing and Supporting Information Technologies

In 2001, IBM launched the Autonomic Computing initiative, a vision striving for system self-management. The idea of autonomic computing originates from the human autonomic nervous system. This system tells the heart how fast to beat, checks the blood's sugar and oxygen level, etc. All of these are done automatically without conscious human attention. That's precisely what is needed to initiate the next era of computing: autonomic computing, also known as self-managing systems. People are aware that it's a paradox that, to achieve such autonomic features, the system must become even more complex by embedding the complexity into the

[2]A data stream is a real-time, continuous, and ordered (implicitly by arrival time or implicitly by time stamp) sequence of items [43]. It is impossible to control the order in which items arrive, nor is it feasible to locally store a stream in its entirety. Queries over data streams run continuously over a period of time and incrementally return new results as new data arrive.

system infrastructure itself, so that the system management can be automated.

Autonomic Computing includes the following four features: self-configuring, self-healing, self-optimizing and self-protecting [72]. Self-configuring involves automated configuration of components and systems following high-level policies. Self-healing can be accomplished by automatically detecting, diagnosing and repairing localized software and hardware problems. Self-optimizing involves self-tuning of service parameters. Self-protecting means that the system automatically defends against malicious attacks or cascading failure. It uses early warnings to anticipate and prevent system-wide failure.

In this chapter, we investigate current information technologies that favor autonomic computing, and thus autonomic web-based simulations.

The architecture of an autonomic computing system is a collection of components called autonomic elements, which encapsulates autonomic managers and managed elements, as shown in Figure 3.1. A managed element can be a hardware resource, such as a CPU, or an application software, such as a database server, or an entire system. An autonomic manager is an agent managing its internal behavior and relationships with other agents according to prescribed policies. System self-management will arise from both interactions among agents and from agents' internal self-management.

IBM has conducted research towards autonomic computing, across all levels of computer management, from hardware to software and some of the work has been published in the IBM Systems Journal [65, 6, 84, 29]. On the hardware level, systems are dynamically reconfigurable and upgradeable, enabling the movement of hardware resources (such as processors, memory and I/O slots) without requiring reboots [65]. On the operating system level, an active operating system allows monitoring code, diagnostic code and function implementations to be dynamically inserted and

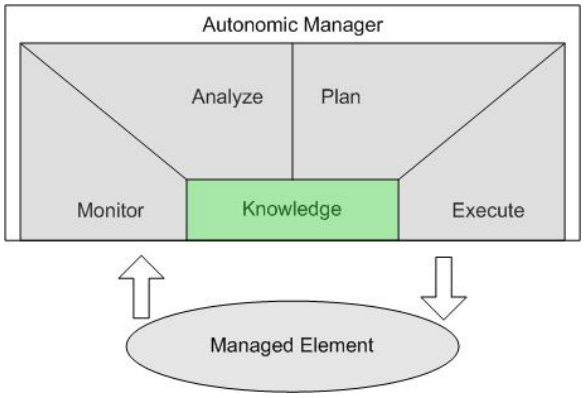

Figure 3.1. An autonomic element consists of a managed resource and an autonomic manager(Adopted from [72])

removed in the system (the so-called "hot-swapping") [6]. On the application level, database self-validate optimizers [84] and web servers are dynamically reconfigured by agents to adapt service performance [29]. In Markl et al [84], an autonomic query optimizer automatically self-validates its model to repair incorrect statistics or cardinality estimates. By monitoring queries as they execute, the autonomic query optimizer compares the optimizer's estimates with the actual cardinality at each step in a *query explain plan*, and computes adjustments to its estimates that may be used during future optimizations of similar queries. In Diao et al [29], the authors use an *AutoTune* agent framework under the *Agent Building and Learning Environment (ABLE)* to automatically tune application-level parameters *MaxClient* and *KeepAlive* to control CPU and memory utilization in an *Apache Web Server*.

From the above examples, we see that some of the ideas from autonomic computing have already been implemented in practice. In this dissertation, we seek to put some of the autonomic computing ideas into the field of web-based simulation. Next, we survey some of the current technologies that favor autonomic computing and thus autonomic web-based simulations.

Many recent advances in IT are in favor of autonomic web-based simulation. These technologies include on-demand computing, utility computing, grid computing, J2SE (Java 2 Standard Edition) 1.5 (also called J2SE 5.0), service-oriented computing, dynamic systems initiative, recovery-oriented computing, externalized architecture adaptation, self-organizing architecture, and workflow management system, to name a few.

On-demand (OD) computing is an increasingly popular enterprise model in which computing resources are made available to users as needed. The resources may be maintained within the user's enterprise, or made available by a service provider. On-demand computing products are rapidly becoming prevalent in the marketplace. Computer Associates, HP, IBM, Microsoft, and Sun Microsystems are among the most prominent on-demand vendors. Concepts such as grid computing, utility computing, autonomic computing and adaptive management seem very similar to the concept of on-demand computing. It's said that on-demand computing is a broad category that includes all the other terms[3]. Utility computing, for example, is an on-demand approach that combines outsourced computing resources and infrastructure management with a usage-based payment structure.

Grid computing is distributed computing whose goal is to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources [36, 35, 64]. In most organizations, there are large amounts of underutilized computing resources. Most desktop machines are busy less than five percent of the time. Even server machines can often be relatively idle. Industry analysts estimate that companies on average utilize only 20% of their server capacity. Grid computing provides a framework for exploiting these underutilized resources and thus has

[3]See http://wikipedia.org/

the possibility of substantially increasing the efficiency of resource utilization. Also, machines may have enormous unused disk drive capacity. Grid computing, more specifically, a *data grid*, can be used to aggregate this unused storage into a large virtual data store, configured to achieve improved efficiency and reliability. Another functionality of grid computing is to better balance resource utilization. For example, grid-enabled applications can be moved to underutilized machines during peak times. In general, a grid can provide a consistent way to balance the loads on CPUs, storages, and other resources. According to the the CIO Today Magazine (March 2004)[4], "Many CIOs are spending good portions of their IT budgets on improving their existing infrastructure". In other words, there is a focus on increasing utilization. Grid computing provides a natural platform for grid-enabled applications, some of which can present autonomic behavior [73, 2]. Therefore, autonomic computing can benefit from grid computing.

Another important advance in information technology is the release of J2SE 1.5[5]. The greatest enhancement of J2SE 1.5 from previous editions is monitoring and manageability. Monitoring and manageability is a key component of RAS (Reliability, Availability, Serviceability) in the Java platform. The release of J2SE 1.5 introduces comprehensive monitoring and management support for the Java platform: instrumentation to observe the Java virtual machine (JVM), Java Management Extensions (JMX) framework and remote access protocols. The JVM Monitoring and Management API specifies a set of instrumentation to allow a running JVM and underlying operating system to be monitored. This information is accessed through JMX MBeans and can be accessed locally within the Java address space or remotely using the JMX remote interface. One of the monitoring features is the low memory detector. JMX MBeans can notify registered listeners when a low memory threshold

[4]CIO Today: http://www.cio-today.com
[5]See http://java.sun.com/j2se/1.5.0

is crossed. The monitoring and manageability enables self-awareness, which is the basis of autonomic computing; hence J2SE 1.5 is a contributing platform to develop autonomic applications, including autonomic web-based simulations. Furthermore, the possibility to efficiently run separate JVMs provides a basis for isolation of Java processes and achieving self-healing by fast restarting of failed or paused processes.

Service-oriented computing (SOC) is the computing paradigm that utilizes services as fundamental elements for developing applications [93]. Services are autonomous platform-independent computational elements that can be described and discovered. Adopting the service-oriented computing paradigm has the potential to reduce programming complexity. However, before the service-oriented computing paradigm becomes reality, there are a number of challenges to be addressed including service modeling and design methodologies, architectural approaches, service deployment and composition, and supporting infrastructure. Service oriented computing models have been developing with technologies such as Corba, EJBs and .Net. With today's level of maturity, these technologies can provide a foundation for autonomic systems. The Forrester report on the Fabric Operating System [105] claims that the Fabric OS will make service-oriented computing a reality, predicts that service-oriented systems will be running enterprise applications before 2007, and provides a roadmap for companies on how to prepare to enable their software systems on the new coming computing platforms.

Analogous to Autonomic Computing, the Dynamic Systems Initiative (DSI) is a Microsoft-led industry effort on Windows systems to address the complexity in today's IT systems and improve their manageability. It has a purpose similar to the vision of Autonomic Computing. The goal of DSI is to provide solutions to simplify and automate all aspects of the application life cycle and how businesses design, deploy and operate distributed systems [88]. Windows Server 2003 marked

the beginning of the DSI product roadmap, providing a foundation for building dynamic systems. Visual Studio .Net 2003 and the Windows Server management tools - Systems Management Server 2003 and Microsoft Operations Manager 2005 - are examples of products that are taking advantage of DSI.

Recovery-Oriented Computing (ROC) [95] emphasizes recovery from failures rather than failure-avoidance, which is a traditional fault-tolerant approach. The ROC hypothesis:"Repair fast to improve dependability and to lower cost of ownership", embraces the fact that hardware faults, software bugs, and human errors are facts to cope with, not problems to be solved. ROC provides new techniques for this purpose: (1) Recovery experiments to test repair mechanisms in development and in the field; (2) Aids for diagnosing the causes of errors in live systems; (3) Partitioning to rapidly recover from faults and to contain them; (4) Reversible systems to handle undo of failed operations and provide a safety margin; (5) Defense in depth in case the first line of defense fails to trap an error; and (6) Redundancy to survive faults and failing fast to reduce MTTR (Mean Time to Repair).

Adaptation of software architecture uses externalized managers to provide self-healing of large systems. For example, IBM's Tivoli monitoring is a systems manager using an expert system to isolate problems and correct them on local machines [79]. The monitoring software maintains an external model which is accessed by scripts that isolate problems and repair faults. Externalized adaptation favors a centralized system organization. In centralized model-based adaptation, an architecture manager maintains, analyzes and corrects the system model. Self-organizing architecture forces constraints on the system components, and the components are responsible for managing themselves[40].

Besides managing systems from the architecture level, the data flow in a system could also be managed. A workflow process is a series of actions that involves

coordination of tasks, handoff of information and synchronization of activities [9]. A workflow management system manages connections between tasks located on different hosts. A workflow management model proposed by Shrivastava supports transactional dynamic reconfiguration of workflow schema and executing workflow instances [111].

Although significant effort has been reported on guaranting high reliability and availability of information systems, little research has been reported on how to guarantee high reliability and availability of web-based simulations. We present the AWS framework to develop and deploy reliable web-based simulations. Autonomic web-based simulations cannot become a reality without satisfying certain requirements.

3.2 Requirements for AWS

We propose the following three basic requirements for AWS:

- The simulation should be able to restart from near the point of failure, since it is too costly to restart from the beginning.
- The simulation should be aware when a failure may occur in the near future so that it can terminate itself gracefully and restart.
- The simulation should be deployed to a self-managing computing infrastructure.

We discuss each requirement in detail and show how they can be achieved.

3.2.1 Requirement 1: Checkpointing and Restarting

One of the basic requirements for AWS is that simulations should be able to restart from near the point of failure. This is accomplished by the mechanism called checkpointing. In the previous chapter, we presented and compared three models to analytically determine the optimal checkpoint interval. The optimal checkpoint interval is the root for the following equation:

$$\frac{x+c}{M} = -\log(1-\frac{x}{M}) \tag{3.1}$$

where M is the mean time before crash and c is the mean checkpoint time. Note that the optimal checkpoint interval is independent of the restart time r. We proved that the above equation 3.1 has one and only one root in the interval $(0, M)$ and the solution can be found numerically by using a simple bisection algorithm.

Since $0 < \frac{x}{M} < 1$, we can expand $\log(1-\frac{x}{M})$ using Taylor series. Equation 3.1 can be written as

$$\frac{c}{M} = \sum_{n=2}^{\infty} \frac{x^n}{nM^n} \tag{3.2}$$

We see that $\frac{c}{M}$ increases monotonely as $\frac{x}{M}$ increases. Vice versa, $\frac{x}{M}$ increases monotonely as $\frac{c}{M}$ increases. From Theorem 2.3 in Chapter 2, the expected total execution time of a simulation increases as $\frac{c}{M}$ increases. Therefore, a simulation can complete more quickly if the checkpoint process can be accomplished faster. Furthermore, from the same theorem, we see that the total execution time can be reduced if restart time r can be reduced and mean time before crash M can be increased. The restart time r includes time spent to detect the failure and to restore checkpoint data. Failure detection time can be dramatically reduced if the failure can be anticipated in advance. This can be achieved by the second requirement for autonomic web-based simulation.

3.2.2 Requirement 2: Proactive Failure Detection with Java Management

Historic concerns about the use of Java for compute-intensive[6] applications are fading as Java is reaching performance parity with other languages (such as Fortran and C/C++) and has begun to be used in scientific simulations [41, 119, 122, 11]. With the release of J2SE 5.0, monitoring and management APIs are added that expose information about the JVM and the underlying operating system. The monitoring and management APIs use Java Management Extension (JMX) to expose data in areas like memory, thread, runtime, operating system and garbage collection [66]. The use of JMX allows the information to be available locally or remotely to applications that support JMX. The performance impact of extracting the JVM information is extremely low [32]. Therefore, it is worthwhile to extract the JVM information for simulation internal management.

Manageable simulations provide mechanisms by which it is possible to monitor, track and control themselves. Monitoring means capturing runtime information from the simulation; tracking means observing aspects of a simulation over a period of time; and control means altering the behavior of a running simulation. The information exposed by the monitoring and management APIs in J2SE 5.0 can be used in:

- External monitoring and management: Allows external monitoring software such as IBM Tivoli and HP OpenView to monitor the JVM and the simulations.
- Internal monitoring and management: Allows the simulation developers to add logic to self-monitor and manage the JVM to make the simulation self-managing.

[6]Compute-intensive is a term that applies to any computer application that demands a lot of computation, such as meteorology programs and other scientific applications. A similar but distinct term, computer-intensive, refers to applications that require a lot of computers, such as grid computing. The two types of applications are not necessarily mutually exclusive: some applications are both compute- and computer-intensive.

Table 3.1

MANAGEMENT INTERFACES FOR MONITORING MEMORY AND CPU

Managed Resource	*Interfaces in java.lang.management*
Memory	MemoryMXBean MemoryPoolMXBean MemoryManagementMXBean RuntimeMXBean GarbageCollectorMXBean
CPU	OperatingSystemMXBean ThreadMXBean RuntimeMXBean

From the point view of simulation developers, memory consumption and CPU usage are the major concerns of monitoring simulations. Table 3.2.2 lists the interfaces in java.lang.management that can be used to monitor memory consumption and CPU usage.

From our experience in developing the NOM simulations, OutofMemoryError is not an uncommon exception. Collection classes such as HashTable and Vector can be easily misused and result in memory leaks. Another possible cause of an OutOfMemoryError exception is the most common unintentional object retention, which causes the heap to grow to an unexpected size. The memory usage can be polled using the getUsage() method, or an event notification-based mechanism can be used to monitor memory by setting a usage threshold using

```
memoryPool.setUsageThreshold(MEMORY_MAX)
```

where memoryPool is an instance of MemoryPoolMXBean and MEMORY_MAX is the peak memory value in bytes. When the memory usage exceeds the threshold, an event MemoryNotificationInfo will be generated. When the event is received, the simulation can checkpoint and terminate gracefully.

Unexpected CPU usage of a running simulation indicates a simulation may not be behaving as expected. The CPU usage information can be obtained from the ThreadMXBean interface by calling methods such as getThreadInfo(), getThreadCpuTime(), and getThreadUserTime(). A running simulation with unexpected CPU usage can terminate itself and return to the simulation job queue (described in later sections).

Adding the monitoring and management code into simulation codes makes running simulations (self-)manageable. Hence, to build autonomic web-based simulations, we require the simulations to be manageable using the monitoring and management APIs.

3.2.3 Requirement 3: self-managing Infrastructure

Web-based simulations must be deployed to the web for use through some computing systems. We propose a self-managing computing grid that enables users to run simulations and obtain simulation reports anytime from anywhere. Figure 3.2 shows the network diagram of the computing grid which serves the simulations.

The firewall/router forwards incoming HTTP traffic to an appropriate application server, which hosts the front-end web applications for user interface and simulation data analysis and visualization. The operational database serves as the back end for the web applications and simulation servers on which simulations run. The data warehouse server is used for efficient data analysis and data mining. Although not shown in the figure, a standby database server and a standby data warehouse are configured so that in case the primary database (data warehouse) is down due to hardware/software errors, the standby database can take the role of the primary database. There is an autonomic agent running on each component of the computing infrastructure to monitor and manage the corresponding component. And an

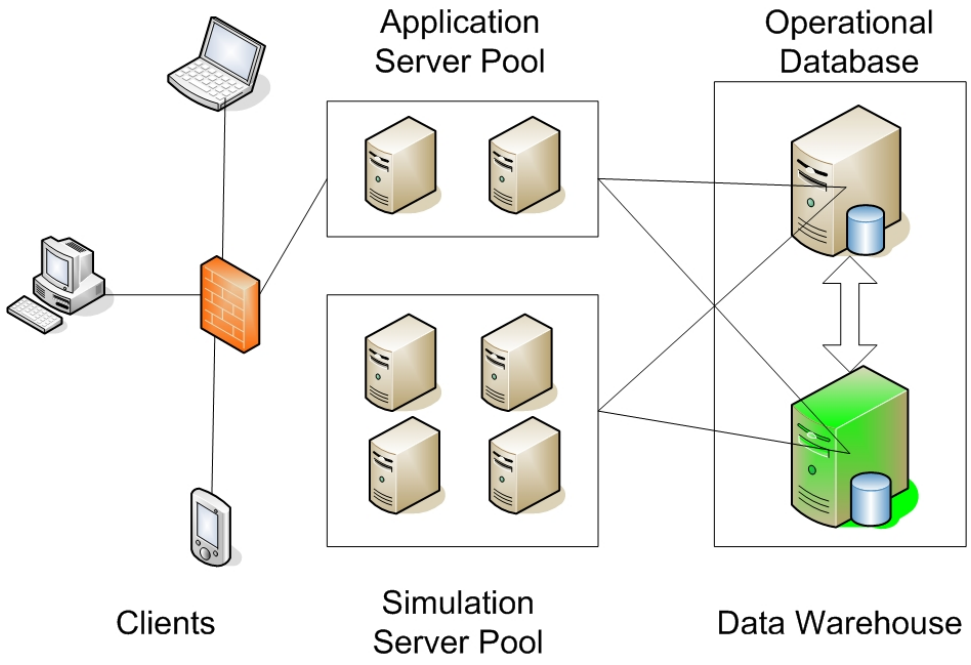

Figure 3.2. Network diagram of the self-managing computing grid: autonomic agents are installed and running on the firewall/router, application servers, simulation servers, database server and data warehouse; an autonomic manager is running inside the database server, which is implemented using stored Java procedures

autonomic manager implemented using stored Java procedure running inside the operational database server. The autonomic manager includes a simulation dispatcher which distributes user submitted simulation jobs and crashed jobs to appropriate simulation severs.

Figure 3.3 shows the data flow diagram[7] for autonomic web-based simulation. When a user submits a simulation, the information related to the simulation is stored in the Submission Status data store. The autonomic manager checks the submission, chooses an appropriate simulation server with the smallest load average to execute the simulation, and puts the submission into a job queue. An autonomic agent on a simulation server checks the job queue and invokes the simulation if any. The autonomic agent also monitors simulations currently running on its corresponding simulation server. If a failure is detected, the submission status is updated and the autonomic manager will redistribute the simulation to another appropriate simulation server.

Self-awareness is a basic feature for autonomic applications, including autonomic web-based simulations. To enable self-awareness of AWS, we need a data model to enable efficient storage and retrieval of information about all components in the system. Figure 3.4 shows the entity-relationship diagram. Note that not all entities and attributes are listed here because of space constraints. Next we describe the ER diagram in more detail.

- **User** (ID, firstname, lastname, password, email, role_id, verified_by, member_since) is the table for all users including administrators of the system.
- **Role** (ID, name, description): User access to the system is controlled by roles; possible roles including admin, normal, none.

[7] A data flow model is also known as a process model. Process modeling is an analysis technique used to capture the flow of inputs through a system to their resulting output. The model is simple in that there are only four types of symbols - process, dataflow, external entity and data store.

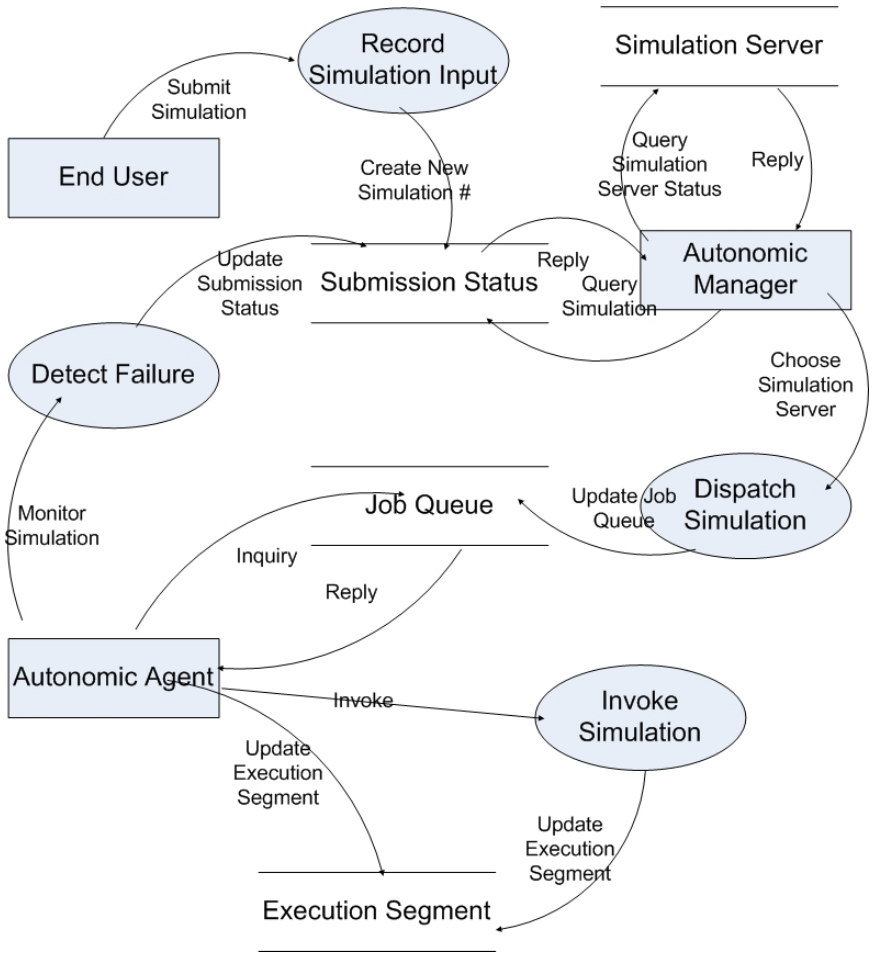

Figure 3.3. Data flow diagram of AWS describing the process after a simulation is submitted

- **Model** (ID, name, description, checkpoint_interval, restart_time, checkpoint_time, MTTF, code) is the table for all simulation programs. MTTF (mean time to fail), checkpoint_interval, restart_time and checkpoint_time are important features of simulation models and are determined by experiments. Code stores the path where the compiled simulation program resides.

- **Input** (simulation_ID, parameter_name, parameter_value) is the input table for instances of simulations.

- **Server** (ID, hostname, IP, DATA_HOME, free_space, loadavg, free_memory,

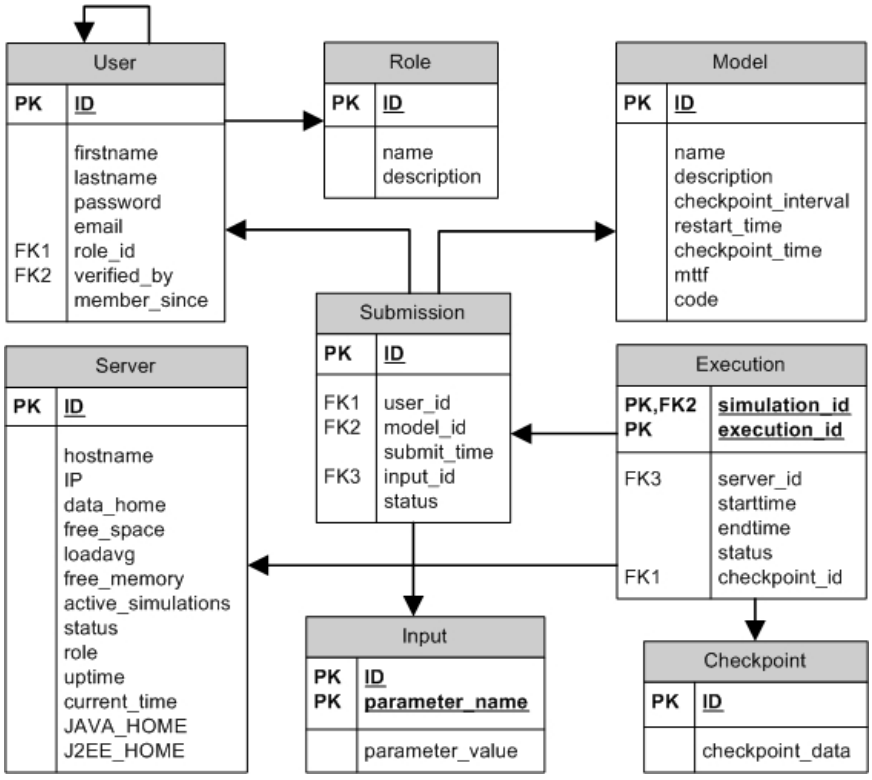

Figure 3.4. ER diagram for self-awareness of AWS components

active_simulations, status, role, uptime, current_time, JAVA_HOME, J2EE_HOME): Available application servers and simulations are stored in this table. Information such as load average, and free memory about these servers is periodically updated by autonomic agents. DATA_HOME is the directory on a simulation server where simulation output files are stored. Based on the percentage of free space available on the simulation server, simulation output files are purged from local drives and stored on a configured NFS server. Role denotes whether a server is currently a simulation server or an application server.

- **Submission** (ID, user_id, model_id, submit_time, input_id, status): Information about user submitted simulations are stored in this table. A submission's status can be submitted, crashed, restarted.

- **Execution** (simulation_id, execution_id, server_id, starttime, endtime, status, checkpoint_id): A simulation may break down into execution segments; each

segment records the start time, end time of the segment, as well as checkpoint data.

- **Checkpoint** (ID, checkpoint_data): Checkpoint data for simulations in each execution segment is stored in this table. Note that this table grows very fast, and therefore, data is frequently purged for better performance.

The efficiency of the above data model is vital in the design of the system, since the information in these tables are frequently queried and updated by the autonomic manager and autonomic agents to direct their behaviors.

3.3 Self-manageability of AWS

The autonomic manager is a Java stored procedure which resides inside the operational database server. Functionalities of the autonomic manager include dispatching simulation jobs to simulation servers, querying the operational database metrics such as free space, memory and CPU usage, managing autonomic agents running on simulation servers, application servers, database servers, data warehousing and the firewall/router. Functionalities of autonomic agents vary on different components. Appendix H shows an implementation of autonomic agents on simulation servers. Self-manageability of autonomic web-based simulations includes self-configuring, self-healing, self-optimizing and self-protecting. We show how these features are achieved in next sections.

3.4 Self-configuring

Self-configuring involves autonomic incorporation of new components and autonomic component adjustments to new conditions. Self-configuring is accomplished by the autonomic agents. Most of the self-configuring tasks are for the purpose of performance optimization, for example, application server self-configuration is for

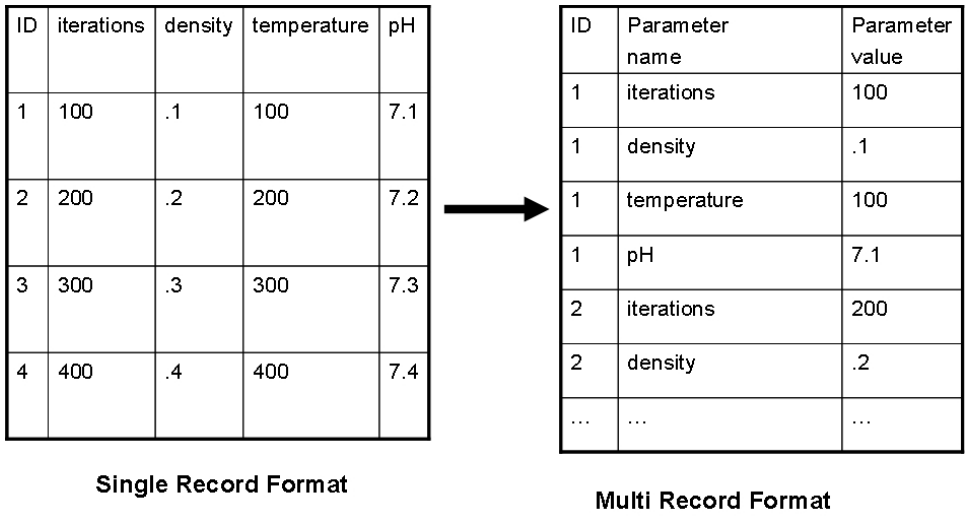

Figure 3.5. Multi record format to solve schema changing problem

best performance. Next we describe a few self-configuring tasks in our implementation.

3.4.1 Data-Driven Web Interface

Unlike business applications, scientific applications often result in schema change. For example, the input metadata for a simulation program may change because of research uncertainty. Based on our experience in developing the NOM simulators [13], the input parameters changed several times because of research needs. Such change results in change of database schema and the corresponding web interfaces. However, the changes can be automated with the help of multi-record data format as shown in Figure 3.5. An advantage of the multi-record data format is that we can use the input table for all simulation models. A data driven interface is designed so that the input metadata is retrieved to avoid manual editing of the JSP(JavaServer Page)/UIX (User Interface XML) code.

3.4.2 Automatic Configuration of the Firewall/Router

Although the web application is installed on all simulation servers inside the computing grid, only one application server is configured to be active at a certain time. The firewall/router forwards incoming HTTP traffic to the application server. In the case that the application server is down, another application server is started by an autonomic agent directed by the autonomic manager. The firewall/router must be re-configured so that the HTTP traffic is routed to the current application server. This is done by an autonomic agent on the firewall/router by issuing an "iptables" command. How to detect the failure of an application server is discussed in the section of self-healing of application servers.

3.4.3 Automatic Configuration of the Simulation Servers

A new simulation server can be discovered if an autonomic agent is installed and running on it. The autonomic agent issues an "insert" into the Server table. Hostname, IP address, and metrics such as load average, free memory and free space can be obtained by calling operating system commands. For a new simulation server, the simulation programs may not have been installed locally. The Model table is queried by the autonomic agent to obtain the path where the code is located on the database server. An "scp" command is issued then to copy the simulation code from the database server. The following code snippet shows the "insert" command for registering a new simulation server.

```
insert into server values (server_id.nextval, hostname, ip, $HOME,
free_space, load_average, free_memory, 0, 'up', 'simserver',
sysdate, sysdate, null, null);
```

Information in the Server table is updated frequently (every 5 seconds in our implementation). When an "update" command is issued, the old record is inserted into the Server_History table by a database trigger automatically. Server_History

has the same structure as the table Server, but its primary key is now the combination of ID and current_time. Data in the Server_History is valuable to analyze the performance of simulation servers. We'll discuss this issue in this chapter and Chapter 4 when dealing with self-optimizing and data warehousing for AWS.

3.5 Application Server Self-configuration

The J2EE application server uses a distributed multi-tiered application model for enterprise applications. Application logic is divided into components according to function, and the various application components that make up a J2EE application are installed on different machines depending on the tier in the multi-tiered J2EE environment to which the application components belongs. Figure 3.6 shows the architecture of multi-tiered J2EE applications. J2EE applications are generally considered to be three-tiered applications because they are distributed over three locations: clients machines, the J2EE server machine (including the web tier and business tier), and the database or legacy machines at the back end. Three-tiered applications extend the standard two-tiered client-server model by placing a multi-threaded application server between the client application and the back-end storage.

Issues such as concurrency, security, transactions, database accesses are handled by the application server. The performance of an application server depends heavily on its configuration. A typical commercial off-the-shelf (COTS) application server has hundreds of performance related parameters that are tunable. Appropriate configuration is a difficult and error-prone task [98, 126, 70] due to the large number of parameters and inter-relationships among them. Such parameters include thread pool sizes, queues, data source connection pool size, dynamic cache size, timeouts and retries, and memory allocations.

As shown in Figure 3.7, enterprise java beans invoked by servlets are managed

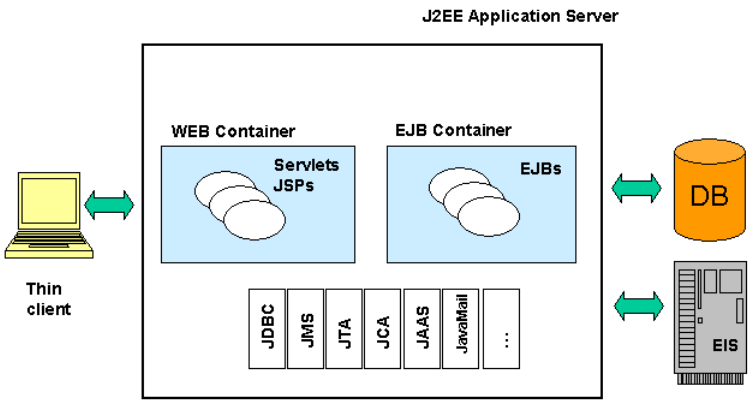

Figure 3.6. Multi-tiered J2EE applications (adopted from Sun Microsystems)

by the Object Request Broker (ORB) inside the EJB container. The ORB thread pool size can be customized. If an EJB request cannot be served by a server thread, it will be put in the waiting queue and the queue size can be tuned also. The size of the data source connection pool affects the number of concurrent accesses to the database. For applications that perform a combination of updates and complex parallel queries to the same database table, performance can be improved by limiting the number of database connections so that contention is reduced.

The performance of the application server depends on these parameters in a non-linear way. Even worse, the performance depends on these parameters in a non-convex way, although our experiments do suggest that the average response time is convex against some parameter[8] by fixing others, as shown in Figure 3.8, which is

[8]A function is convex on each parameter may not be convex itself. For example, the function $f(x, y) = x^2 + 4xy + y^2$ is convex for x (or y) by fixing y (or x) since $\frac{d^2 f}{dx^2} = 2 > 0$ and $\frac{d^2 f}{dy^2} = 2 > 0$.

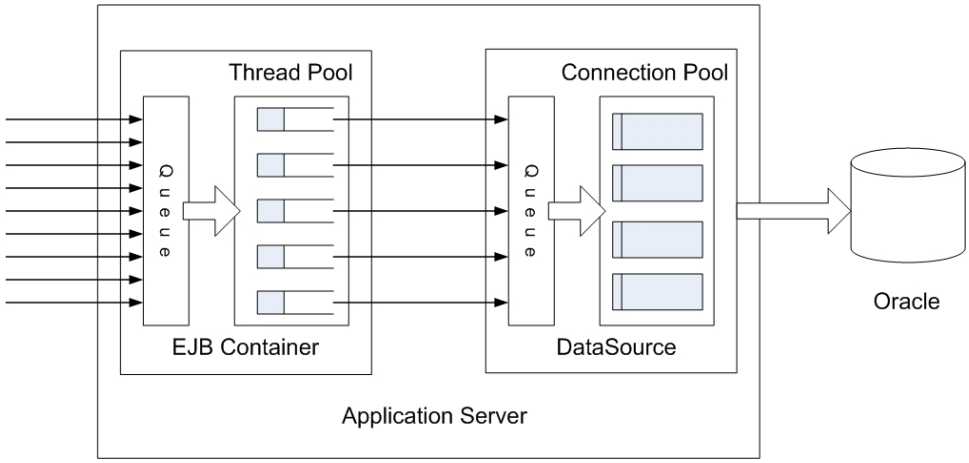

Figure 3.7. Self-tuning thread pool size and connection pool size

obtained from experiments. Non-convexity suggests that the global optimum is not easy to find. Furthermore, once these parameters are set, it is usually not possible to change them without shutting down the application server.

In today's practice, configuration of application servers is done in a trial-and-error manner, which needs a significant amount of expert knowledge about the application server and applications. Besides, the trial-and-error process requires a significant amount of time to do load testing. Therefore, an automatic configuration of the application server by an autonomic agent is useful in practice. In this section, we formulate the problem to be a global optimization problem. Similar work on configuring large-scale networks can be found in [117, 126].

3.5.1 Problem Formulation

The application server parameter configuration problem can be formulated as follows:

Given a set of tunable parameters $x_1, \cdots, x_n$ where $x_i \in I_i = [x_{min}^{(i)}, x_{max}^{(i)}]$ for

However, $f(x, y)$ is not convex since its Hession matrix $\begin{pmatrix} 2 & 4 \\ 4 & 2 \end{pmatrix}$ is not positive definite.

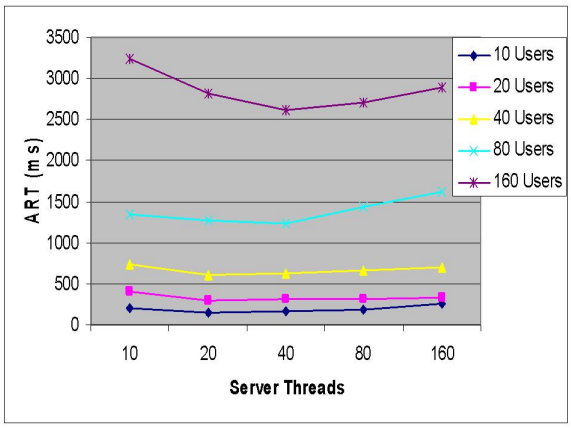

Figure 3.8. Average response time versus server thread pool size

$1 \leq i \leq n$, and a continuous objective function $f : I \rightarrow R$ where $I = I_1 \times \cdots \times I_n$ is compact in R^n, find a global minimum,

$$x^* = \arg\min_{x \in I} f(x). \tag{3.3}$$

Since f is continuous on a compact set I, it is guaranteed that the global minimum and global maximum exist. The objective function f can be the measure of the average response time (ART), or the system throughput - transactions per second (TPS), or a combination of both. In practice, the objective function f is analytically unknown (i.e., it is not possible to explicitly express the objective function f) and its function value at a certain parameter setting can only be obtained through experiments.

It is generally hard to find the global optimum, therefore, in practice, a *good* solution near the optimum is acceptable for efficiency reasons. There are a number of heuristic methods for global optimizations such as recursive random search [117], smart hill climbing [126], simulated annealing [101], Tabu search [42], and genetic algorithms [44]. We review some of these algorithms. Recursive random search

utilizes pure random sampling to first identify promising areas and then to start random sampling processes in these areas that shrink gradually to local optima. The algorithm then restarts the random sampling processes and tries to find a more promising area to repeat the local recursive search. The smart hill climbing algorithm is based on the ideas of importance sampling and Latin Hypercube Sampling. Simulated annealing mimics the physical annealing process, where a solid material is first melted by heating to a very high temperature and then cooled down at a very slow speed to a final crystalling state with the lowest energe.

We use the recursive random search algorithm with slight modification. Define

$$\phi_I(y) = \frac{m(\{x \in I | f(x) < y\})}{m(I)} \tag{3.4}$$

where m is the Lebesgue measure in R^n. Let $y_{min} = \min f(x)$ and $y_{max} = \max f(x)$, then

$$\phi_I(y) : [y_{min}, y_{max}] \to [0, 1] \tag{3.5}$$

is continuous and monotonely increasing. For any $r \in [0, 1]$, there exist $y_r \in [y_{min}, y_{max}]$ such that $\phi_I(y_r) = r$. Define

$$A_I(r) = \{x \in I | f(x) < y_r\}. \tag{3.6}$$

Then $m(A_I(r)) = r \cdot m(I)$ and $\lim_{r \to 0} A_I(r) = \{x^*\}$.

Let $x_i, i = 1, \cdots, m$ be a sequence of random samples, and let $x_0 = \arg \min_{1 \le i \le m} f(x_i)$, then the probability that $x_0 \in A_I(r)$ is

$$p(x_0 \in A_I(r)) = 1 - (1 - r)^m. \tag{3.7}$$

For any $0 < r < 1$, the above probability tends to 1 with increasing m. For example, let $r = 0.1$ and $m = 44$, then the above probability is greater than 0.99.

In other words, it takes only 44 samples to reach a point in the promising area $A_i(0.1)$ with probability 0.99. The recursive random research algorithm makes use of this property to gradually shrink the promising area to reach a minimum as follows: when x_0 was found, a neighborhood of x_0, $N_D(x_0) = \{x \in D | |x_i - x_{0i}| < r^{\frac{1}{n}} |r_i - l_i|, \forall i\}$ where D is initially I and later the new smaller sample parameter space, and r_i and l_i are the endpoints of the i^{th} coordinate.

3.5.2 Configuring the Application Server

The performance related parameters are tunable in the application server configuration files *server.xml* and *data-sources.xml*. In our implementation, we configure the parameters *global-thread-pool*, *queue* and *max-connections*, where thread pool size is between 1 and 200, queue size is between 1 and 200, and max-connections is between 1 and 200. Note that max-connections should be bounded by the database initialization parameter *processes*, which was configured to be 300. For a specific parameter setting, we use Grinder [49] to simulate HTTP requests to the application server. The Grinder is an open source load-testing framework available on SourceForge.net. The Grinder can simulate simultaneous clients access the web application using Java threads. The Grinder calculates the average response time when all transactions are completed. There are many other load testing tools, including JMeter[9], Cactus[10] and http_load[11].

We implemented the recursive random search algorithm to tune the aforementioned parameters based on the simulated load of 50 concurrent users. The result is shown in Table 3.2.

Note that the connection pool size 30 is less than the thread pool size, which confirms that to obtain ideal performance, the server thread pool and data source

[9]See http://jakarta.apache.org/jmeter/
[10]See http://jakarta.apache.org/cactus/
[11]http://www.acme.com/software/http_load/

Table 3.2

OPTIMAL CHOICE OF THREAD POOL SIZE, QUEUE SIZE AND CONNECTION POOL SIZE

Thread Pool Size	Queue Size	Connection Pool Size
40	77	30

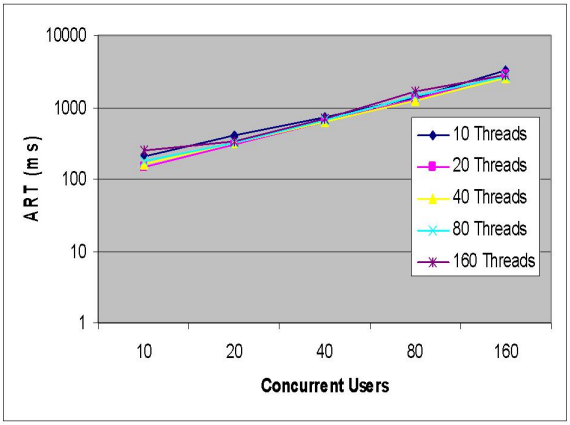

Figure 3.9. Average response time versus number of concurrent users and for each server thread setting

connection pool should have a threading "funnel effect" [70]. In other words, the data source connection pool size should be less than the server thread pool size. Once these parameters are set, the average response time is linearly dependent on the number of concurrent users, as shown in Figure 3.9.

3.6 Self-healing

Self-healing can be accomplished by automatically detecting, diagnosing and repairing localized software or hardware problems. Some sort of redundancy is necessary to achieve self-healing. For example, the operational database is vital in the

successful operation of the system, therefore, a hot standby database is configured so that it can take the role of the primary database in case of failure. The application servers and simulation servers are also configured with redundancy. Failures of local running simulations and web applications are detected by the autonomic agents running on simulation servers and application servers. Failures of simulation servers and application servers are detected by the autonomic manager in which the autonomic agents do not respond in a timely fashion.

The non-functional period of a failed service or node is comprised of two distinct phases: periods when a system is unaware of a failure (failure-detection latency) and periods when a system attempts to recover (failure-recovery latency) [27]. For example, restart time of a simulation consists of a failure detection phase and a redo phase. As we learned from the previous chapter, the predicted total execution time of a simulation is exponentially proportional to the restart time. Therefore, a fast failure detection algorithm dramatically improves the total execution time of a simulation. With the Java Monitoring and Management APIs used in simulation programs, the failure detection phase can be expedited.

3.6.1 Self-healing Application Servers

Application servers are middleware running J2EE applications. Although commercial application servers are tested against failures, software errors still occur. For example, OutOfMemoryError exception is not uncommon for commercial application servers. A search on "application server OutOfMemoryError" on Google returns thousands of hits. Usually the number of such exceptions can be reduced by initiating a larger heap size with the -Xmx option. However, this error still occurs. Figure 3.10 shows an instance of the OutOfMemoryError in our application log. We take the following actions for the purpose of self-healing application servers.

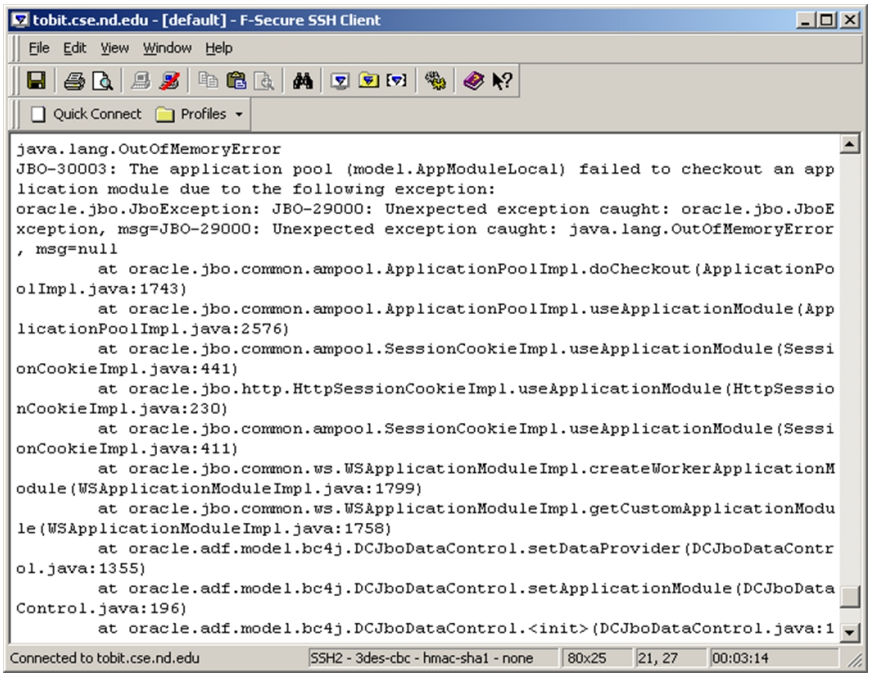

Figure 3.10. An instance of OutOfMemoryError in application log

- Although JMX is not available in Oracle Application Server, we can use a similar tool called dynamic management service (DMS)[12] to monitor an running application server instance. DMS can monitor the total and used heap memory. Once a usage threshold is crossed, the application server is restarted.

```
if(usage.getUsed()/usage.getMax() > SOME_THRESHOLD){
  shutdown_app_server();
}
```

  where SOME_THRESHOLD is the ratio of the amount of used memory over maximum amount of memory. If the threshold is crossed, another application server will be started by a local autonomic agent directed by the autonomic manager, and the application server is shutdown gracefully by the local autonomic agent. And the autonomic agent on the firewall/router will issue an "iptables" command to forward HTTP traffic to the new application server.

[12]See http://download-west.oracle.com/docs/cd/B10464_01/core.904/b12020/toc.htm

- Monitoring the application server by the autonomic agent on the firewall/router by using the "wget" command as follows:

```
#!/bin/sh

( wget -1 -s -O - $AS_SERVER &
  as_pid=$!
  echo $as_pid > $AS_PID
  {sleep 2; kill $as_pid; } &
) | head -1 | cut -d' ' -f2 > $AS_RESPONSE
resp=`cat $AS_RESPONSE`
if [ $resp == "200" ]; then
  echo 1 > $ASUPDOWN
else
  echo 0 > $ASUPDOWN
fi
```

  If the return code of the command is not 200, then the application server is believed to be down. The monitoring script may hang if the application server is not reachable, therefore, we kill the script after 2 seconds. In case the application server is down, the local autonomic agent notifies the autonomic manager by updating the corresponding entry in the Server table and another application server will start by the corresponding local autonomic agent.

3.6.2 Self-healing Simulation Servers

The autonomic agents on the simulation servers update the corresponding entry in the Server table frequently (every 5 seconds in our implementation). This operation serves as a heart-beat of the simulation server. A simulation server is considered to be down if the update operation is not accomplished in a timely fashion. More precisely, if $sysdate - current\_time > \frac{5}{24*60*60}$, the simulation server is considered to be down. All simulations currently running on the simulation server are marked as failed. Then the entries in the Submission table and Execution table are updated. The autonomic manager will dispatch failed simulations to appropriate simulation

servers. Local autonomic agents will restart the failed simulations by querying the Execution table and by fetching checkpoint data from the Checkpoint table.

3.6.3 Self-healing Failed Simulations

As described before, the Java Monitoring and Management APIs can be used to monitor memory usage and CPU usage. Once these monitored metrics exceed thresholds, the simulation terminates gracefully and corresponding entries in the Submission and Execution tables are updated. Exactly the same as described in the above section. The local autonomic agents monitor running simulations as through the MBean interface in the simulation. If a response does not arrive in a timely fashion, the execution segment of the simulation is marked as failed and a "kill" command is issued in case the process is running but not responding. Failed execution segments result in modification of the Submission and Execution tables the same way as described above.

An execution of a simulation is monitored as follows: the autonomic agent retrieves status (EXECUTING or COMPLETED) and pid from the EXECUTIONS table for its underlying simulation server. If the status of an execution is EXECUTING, it then queries the underlying operating system to see whether the simulation is running with the corresponding pid. If such pid does not exist, the execution is reported as CRASHED.

3.6.4 Self-healing Database Servers

The health of the operational database is vital in the success of the AWS system. Therefore, we have done a large amount of work to monitor the operations of the database server, as listed in Table 3.3. The autonomic agents residing on the database server and an external machine (currently, joy.cse.nd.edu) monitor the managed resources, analyze the exceptions, plan and execute corresponding actions. Appen-

dix C shows a script to monitor the availability of application server and database server.

Table 3.3. DATABASE SERVER MONITORING

Managed Resource	**Monitor/Analyze/Plan/Execute**
Database connection	To monitor the availability of database server and listener. If testing connection fails, the listener log file is deleted and listener is restarted
Alert logs	To check the alert log file periodically to identify any logs for error or warning. If fatal errors are detected, a message is sent to the administrator. Fatal errors are difficult to be handled by the autonomic agent. Therefore, the database administrator get involved to solve the solution.
Tablespace size	To check the size of tablespace and free space available. If the tablespace reaches the threshold limit, it will automatically extend the tablespace and send a message to the administrator.
Archive logs	To check if the archive logs are created properly in the current directory and check whether the archive logs are backed and applied properly to the standby database server. If not, a message is sent to the administrator.

Table 3.3. *CONTINUED*

Managed Resource	**Monitor/Analyze/Plan/Execute**
Hit ratio	To check the overall buffer cache hit ratio for the entire instance since it was started. If the ratio reaches the threshold limit, certain actions are taken such as dynamically adjusting memory (for example, the shard_pool_size) and a message is sent to the administrator.
Listener log file size	To check the size of listener log files on each databases, including standby databases. If the listener log file size reaches the threshold limit, the listener logs files are emptied and a message is sent to the administrator.
Size of extents	To check the segments of each tablespace, number of extents, initial size of extent, next extent size, minimum extents, maximum extents and status. Certain actions are taken to avoid the infamous ORA-00555: rollback segments too old error.
Disk space	To check the disk space availability for data files, log files and temp space. If the available size reaches the threshold limit, certain actions such as purging and compressing files are taken, and a message is sent to the administrator.
CPU usage	To monitor the CPU performance and its usage, and log the metrics to the database.

Table 3.3. *CONTINUED*

Managed Resource	**Monitor/Analyze/Plan/Execute**
Memory usage	To monitor the memory usage, memory page faults and log the metrics to the database.
Resource locks	To check locked resources and logs to the database
Invalid objects	To identify invalid objects in the databases. Invalid objects are recompiled by the agent by issuing "alter object_name compile" SQL command.
Chained rows	To identify chained rows. And a batch operation is done to relocate data.

3.7 Self-optimizing

The idea of formulating self-configuring tasks to be global optmizations also applies to self-optimizing. However, usually the performance related parameters cannot be changed dynamically without rebooting the services, which makes self-optimizing technically difficult. For self-optimizing of the AWS system, we would like to address load balancing of simulation servers as a case study of system self-optimizing.

3.7.1 Self-optimizing Simulation Servers

Load balancing simulation servers are achieved in the following manner:

- A new or failed simulation is assigned to a simulation server with lowest load average that is below a present threshold by the autonomic manager.

- The local autonomic agent checks the job queue and starts the simulation if any.
- The autonomic manager monitors the workload of all simulation servers and may request a migration of simulation from a congested server to an under utilized server.

The purpose of simulation migration is to balance load averages and improve simulation completion time. However, excessive migration will cause simulations bouncing back and force between simulation servers, like thrashing in distibuted systems. Then when should a migration occur? It is determined by the following procedure:

- The load averages of the n simulation servers in the last k periods are queried from the Server_History table. Denote them by L_{ij} where $1 \leq i \leq n$ and $1 \leq j \leq k$.
- The expected load average of a simulation server at time period j is $E_{ij} = \frac{\sum_{i=1}^{n} L_{ij}}{n}$.
- A chi-square test is conducted to determine whether there is a bias in the load averages. In other words, whether the simulation servers are not balanced. If the null hypothesis is rejected, then a simulation is migrated from a simulation server with highest load average to another simulation server with lowest load average. The migration takes place in a sequence of actions: first the simulation checkpoints and terminates, then it is restarted in the target simulation server by retrieving checkpoint data from the Checkpoint table.

We have implemented the chi-square test using SQL and typically chosen $k = 21$. In our chi-square test, the degree of freedom $df = 20 * (n-1)$, where n is the number of simulation servers in the last $k = 21$ time periods. Note that a time period consists of 5 seconds in our implementation. The following SQL code snippet shows

the calculation of χ^2 value, and it is highly optimized using Oracle data warehousing analysis functions.

```
with
recent21 as
    (select * from
     (select id, loadavg,
             rank() over (order by current_time desc) as my_rank
        from server_history
       group by id
     )
     where default_rank <= 21
    ),
avg_host_ld as
    (select avg(loadavg) as avg_ld, my_rank from recent21
      group by my_rank
    )
select sum(vv) as chi_square from
    (select (loadavg-avg_ld)*(loadavg-avg_ld)/avg_ld as vv
       from recent21, avg_host_ld
      where recent21.my_rank=avg_host_ld.my_rank
    )
/
```

Note that *df* in our case is usually large, therefore, the chi-square distribution approaches to a normal distribution. Choosing the confidence level 0.05, we see that the critical value is 1.6449. Therefore, if the calculated χ^2 value is above 1.6449, the null hypothesis is rejected, which results in a simulation migration.

3.8 Self-protecting

Self-protecting means the system automatically defends against malicious attacks or cascading failures. It uses earlier warnings to anticipate and prevent system wide failure. In our implementation, access to the system is controlled using different levels of users roles. More precisely, the following actions were taken to protect the system:

- A firewall script was designed and configured so that only one port (80) was open to the public so that the system can be accessed using a web browser.
- Users must register and be verified by the administrator so that access rights can be given.
- Users are given different levels of roles, such as admin, normal and none. Users with the admin role can manage all the components of the system. Users with normal role can only submit simulations and obtain simulation results. Users with the none role cannot access the system, which is the default when a user is created.
- Early warnings such as the prediction of OutOfMemoryError were used to terminate appropriate components such as application servers and running simulations and restart them in a timely fashion to prevent system wide failures.

3.9 Summary

In this chapter, we have presented the framework of autonomic web-based simulations and its prototype implementation. AWS cannot be achieved without satisfying certain requirements. These requirements include

- Ability to checkpoint and restart simulations.
- Application of J2SE 5.0 Monitoring and Management APIs.
- A self-managing computing grid to host simulations.

Autonomic web-based simulations enable end users to run simulations anytime anywhere with guaranteed success. In the next chapter, we design a data warehouse for autonomic web-based simulations. The main purpose of the system is to generate reports about the reliability and performance of the computing grid that hosts web-based scientific simulations.

CHAPTER 4

A DATA WAREHOUSE FOR AUTONOMIC WEB-BASED SIMULATION

Web-based simulation is the integration of the Web and the field of simulation. Autonomic Web-based simulation (AWS)[58] is a framework to develop and deploy reliable web-based simulation based on the vision of autonomic computing[72]. In this chapter, we describe the design and implementation of a data warehouse for autonomic web-based simulation. The main purpose of the system is to generate reports about the reliability and performance of the computing grid that hosts web-based scientific simulations.[1]

4.1 Introduction

Scientific simulations have been increasingly applied to solve a variety of scientific simulations. Domains such as environmental science, in particular, benefit from this capability. For example, we have developed a series of computer models to simulate the behavior of natural organic matter (NOM). These models can play a critical role in understanding and predicting the properties of NOM over time as it evolves from precursor molecules to eventual mineralization [13]. Web-accessible models are developed and deployed so that our collaborators can run simulations and generate reports remotely. However, these scientific simulations are large programs which,

[1]Part of this chapter will appear in Cybernatics and Information Technologies, Systems and Applications 2005 [59]

despite careful debugging and testing, will probably contain errors when deployed to the web for use. Based on the assumption that such errors do exist and the underlying computing system does fail due to hardware/software errors, we proposed a framework called autonomic web-based simulation (AWS) to develop and deploy web-based simulations based on the vision of autonomic computing.

AWS employs a self-manageable computing grid to host the simulation models. The computing grid is based on a multi-tiered architecture that consists of database servers, simulation servers and web servers. One of the key features of the self-manageable computing grid is self-awareness. To enable self-awareness of the system, a normalized data model was designed so that information about the grid was stored in an operational database and trackable at any time. We are interested in collecting information about the grid, and then identifying usage patterns to make better use of resources, which eventually can be used by the system for self-optimization. Therefore, we are in need of a system that can automatically perform data analysis, and trends prediction. Data warehousing is a necessary technology for collecting information from distributed sources including operational databases and then performing data analysis [18, 63, 74, 97].

A data warehouse is generally accepted as a large database that is populated with a significant amount of transactional data. Data warehousing is supposed to provide data that is subject-oriented, integrated, non-volatile and time-variant. The purpose of data is to support scientific discovery, engineering design and decision making with the aid of data analysis techniques. Data mining techniques are typically employed to derive information from the warehouse that will ultimately help in the discovery, design and decision making process. A major difference between a data warehouse and a transaction database is the frequency in which the information is updated. Each new transaction in a transactional database results in the database being

updated. This differs from the data warehousing scenario where data is updated at regular time intervals. Once data is entered into the data warehouse, it is considered to be non-volatile or read only.

4.2 Background and Related Work

During the past years, the database community has been active in modeling and managing high dimensional scientific data. For example, Malon and May [83] describe data models to support data generated from physical experiments. Chen et al [20] discuss the feasibility of building data stream systems for online analytical processing (OLAP). Abdulla et al [1] compare scientific simulation data with data streams and argues that simulation data is a special case of data streams. Based on this argument, they builds the AQSim (Ad-hoc Queries for Simulation data) system. Critchlow et al [25] develop an interface for a data warehouse through a web-based GUI and let users query data. It uses a Java back end with JDBC connection to translate user commands into SQL queries and return the results to the users. Wong et al [124] present a data warehouse framework which encompasses imaging and non-imaging information in supporting disease management and research. The implementation is based on a Java CORBA and web-based architecture. Singhal [112] describes the design of a data warehouse for AT&T business service so that reports about the performance and reliability of network can be generated.

4.3 Data Warehouse for AWS

The purpose of the AWS data warehouse is to generate reports about the performance and reliability of simulation models and the underlying computing grid, and then use them to make better use of the system resources. The computing grid is used to let end users (scientists) run simulations and obtain simulation reports remotely. The data source of the AWS data warehouse is the operational database that stores detailed information about

- Execution segments of simulations. Simulations will probably break down into execution segments because of possible failures. Information related to an execution segments include start time and end time, restart and checkpoint time, etc.

- Capacity and number of connections to the database servers; Data generated from the simulations are stored in local drives and loaded into the database by autonomic agents on simulation servers. The capacity and total connections to the database servers limit the total number of concurrent database accesses.

- Number of concurrent connections to the application servers. This information will be used to guide the application server to spawn an appropriate number of server threads so that user requests can be handled properly.

- Load metric and number of active simulations of the simulation servers. This information is used by the autonomic manager and the simulation dispatcher so that a new or failed simulation is distributed to an appropriate simulation server.

For further details, please refer to Huang et al [60].

We are interested in deriving the following information about simulation models from the data warehouse so that the optimal checkpoint interval can be calculated and the completion time of simulations can be predicted (Interested readers please refer to http://www.nd.edu/~nom/publications/ckpt.pdf for details):

- Mean Time to Crash (MTTC): The average time before simulations crash.

- Mean Restart Time (MRT): The average time to store data from the most recent checkpoint.

- Mean Checkpoint Time (MCT): The average time to write checkpoint data to files or databases.

The main reason to separate decision making data and the operational data is performance. The operational database was designed for transactional workloads. Complex queries and reports degrade the performance of the transactional database.

Furthermore, multi-dimensional modeling is required to optimize complex queries and the report generation process.

The main components of the AWS data warehouse system are as follows.

1. The AWS operational data source contains information about: simulation servers, simulation models, user submissions, and execution segments.
2. The Oracle9i Data Warehouse Builder is an Extract-Transform-Load (ETL) tool that defines how data is extracted from the operational data sources, transformed by the mapping operators, and loaded into the target schema in the AWS data warehouse.
3. Once the AWS data warehouse has been populated, OLAP tools such as Oracle OLAP is configured to access the data warehouse in order to generate reports.

The simple architecture of the AWS data warehouse is shown in Figure 4.1. The following lists some of the advantages of the AWS data warehouse system:

- The separation of decision making data and operational data makes report generation more efficient. Also, the report generation process does not impact the workloads of the transactional database.
- One of the most common queries is the star query in which each of the dimension tables is joined to the fact table using the primary-key/foreign key relationship. Creating appropriate indexes on the dimension tables and bitmap indexes on the fact tables make it possible for the query to use star transformation, an algorithm for optimizing star queries.
- Fact tables and related indexes are partitioned according to list of simulation models and range of simulation IDs. Partitioning the data and indexes makes it possible for data management operations including queries to be performed at the partition level.
- Query performance is further improved by pre-calculating and storing results of frequent queries, especially for queries involving aggregation.

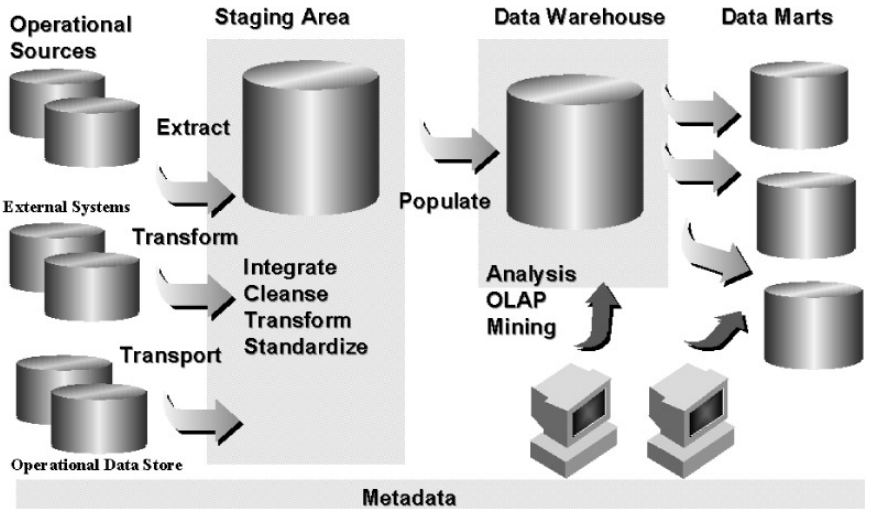

Figure 4.1. Simple Architecture of the AWS Data Warehouse System

4.4 Data Modeling with Star Schema

The execution segments of simulations are modeled using star schema. A star schema is a relational schema whose design represents a multidimensional data model, which consists of at least one fact table and one dimension table. Figure 4.2 shows a star schema of our application, which has a fact table EXECUTIONS and multiple dimensions such as *model*, *user*, *server*, *simulation* and *time*. *Simulation* refers to which simulation the execution segment belongs to. *User* refers to the owner of the execution segment. *Model* refers to the simulation model of the execution segment. *Server* refers to the simulation server on which the execution segment was executing. *Time* represents the time when the execution segment started. The time dimension has a hierarchy that can be used to create aggregates by rolling up or drilling down.

4.4.1 Designing Mappings

Mappings describe a series of operations that extract data from the operational data source, transform it and load it into the target schema. The Oracle Warehouse Builder provides a visual presentation of data flow and operations performed on the data. The Mapping Editor is the interface for designing and editing mappings, which includes a variety of operators to transform data before loading into the target. To improve the performance of reports generation, we extensively used the Aggregator Operator that calculates data aggregations such as summations and averages and provides an output row set with the aggregated data. Examples of the aggregate data include mean time to crash, mean restart time and mean checkpoint time.

The Filter Operator is also used to manage data quality. The filter operator creates a set of conditions that check for rules that data must obey before being loaded into the target schema of the data warehouse. Although most of the data in the operational data source is generated automatically, we still check the validity of the data using the following rules:

- For an execution segment, the end time must be greater than the start time.
- The restart time and checkpoint must be in a reasonable range.

4.4.2 Metadata Management

Metadata is the key success factor of a data warehouse project. It captures all kinds of information necessary to extract, transform and load data from the source systems into the data warehouse; and afterwards to use and interpret the data warehouse contents. Our solution uses the Oracle Warehouse Builder to browse and manage metadata of the AWS data warehouse. The Warehouse Builder Design Repository, installed in an Oracle database, stores the metadata definitions for all of the objects used, including logical and physical schemas, transformation and mapping rules. The warehouse builder browser can be used to view reports of metadata.

These reports include implementation reports, lineage and impact analysis reports and diagrams.

4.4.3 Data Refreshing

The Warehouse Builder can create jobs in the Enterprise Manager so that data loading and refreshing jobs can run at scheduled times. Our experience shows that the parition scheme of tables and indexes is crucial in determining the efficiency of refresh operations during the load process.

Since the source of the data warehouse is a transactional database, we use DML (Data Manipulation Language) statements to refresh data. The data that is extracted from the source is not simply a list of new records to be inserted into the data warehouse. Instead, the new data set is a combination of new records and updated records. In this case, we use the merge operation in Oracle9i, which can be executed using one SQL statement.

The success of data loading and refreshing is monitored by the enterprise manager. The data is automatically reloaded if a failure occurs.

4.5 Status Report

We can query the mean crash time before crash (M), mean restart time (R) and mean checkpoint time (C) for each simulation models. These metrics can be used to calculate an optimal checkpoint interval and predict the completion time of simulations. More precisely, The optimal checkpoint interval is the solution (x^*) of the following equation

$$\frac{x+C}{M} = -\log(1-\frac{x}{M}) \tag{4.1}$$

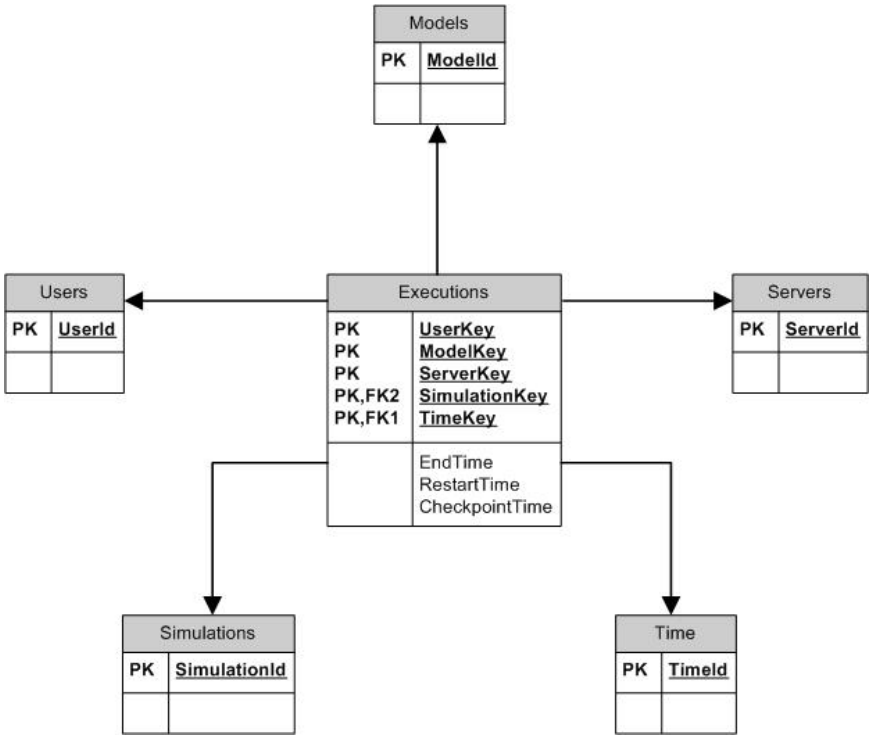

Figure 4.2. A Star Schema in AWS DW

The equation can be solved numerically using a simple bisection algorithm. And the predicted completion time is

$$Ne^{\frac{R}{M}}\frac{1+\frac{R}{M}}{1-\frac{x^*}{M}} \tag{4.2}$$

The following SQL code snippet shows the calculation of the optimal checkpoint interval in PL/SQL:

```
create or replace function
  compute_ckpt_interval (m number, c number, epsilon number := 0.0001)
return number
is
lo number;
hi number;
mi number;
begin
lo := epsilon;
hi := m - epsilon;
  loop
    mi := (lo+hi)/2;
    if first_derivative(lo, m, c) * first_derivative(mi, m, c) > 0 then
       lo := mi;
    else
       hi := mi;
    end if;
    exit when hi - lo < epsilon;
  end loop;
  return mi;
end;
/

create or replace function first_derivative(x number, m number, c number)
return number
is
begin
 return (x+c)/m + ln(1-x/m);
end;
/
```

4.6 Summary

In this chapter, we described the design of a data warehouse that is used to support autonomic web-based simulation. The data analysis capability of the data warehouse provides trends prediction so that it can be used to make better use of the system resources.

The power of scientific simulation is to model a real system and analyze the results. Besides statistical analysis, more advanced analysis techniques are necessary for better understanding and use of simulation data. In the following chapters, we present a general data cleansing algorithm and describe our experience building scientific data warehouses and applying data mining to simulation data.

CHAPTER 5

DATA CLEANSING THROUGH APPROXIMATE STRING JOINS

In this chapter, we describe a novel three step approach data cleansing algorithm through approximate string joins. The goal of the approximate string join is to find (almost) all pairs of strings such that their distance is below a certain threshold. We first map the database of strings into points in an Euclidean space, then use approximate matrix multiplication to find approximately all pairs of close points. These pairs of close points are further evaluated using the string distance to filter false positives. We implement this method in a commercial database. Experiments show that this approach is both effective and efficient. We also point out the limitations of this approach and present the extension of the mapping approach using edit distance with moves.[1]

5.1 Introduction

Data is one of the most valuable components in scientific and engineering research. Often string matching or approximate string match are required in scientific data analysis, e.g., bioinformatics. Data has also become one of the most valuable assets of corporations. It is used in business intelligence applications to support strategy analysis and decision making. But the value of data is highly dependent on

[1] A shorter version appeared in WWW 2004 [56], and a full version is under review with International Journal of Computer Science and Applications

its quality. Poor quality data can cause wrong decision making and even business failure. Thus, it is critical to cleanse data before loading it into data warehouses and using it for data analysis and mining [85].

Data cleansing has attracted much attention from researchers. Real world data may come from heterogeneous sources and may be dirty due to user input errors, different flavors of abbreviations, etc. It is highly likely that two or more records whose textual representation differ may actually represent the same real world entity. For example, the two records "University of Notre Dame" and "Notre Dame University", actually represent the same entity (although their edit distance[2] is 20). Such records are called approximate duplicates. Data cleansing seeks to identify such duplicates and merge them into integrated database records.

Some work has been done to address the problem of approximate duplicates detection, which was also referred to as the merge/purge problem [51] and record linkage [68]. Many data cleansing algorithms are implemented using string join, or text join. Normally, string edit distance, q-gram distance or the vector cosine similarity [47] are used to determine whether two records are "close" enough to be approximate duplicates. More recently, some researchers have used machine learning to "learn" string similarity measures that combine these standard metrics [23, 103, 118, 8].

No matter how the string distance is defined, it is necessary to identify (almost) all pairs of strings whose distances are below a certain threshold. This operation is called string join [46] or text join [78]. Since more and more web sites utilize RDBMS as the backend, it is natural that the data we need to operate is inside some commercial database systems. Therefore, for robustness and efficiency, it is desirable to implement such operations inside the databases.

[2]See Appendix D for an implementation of edit distance in PL/SQL

A baseline approach for string join is the pairwise comparison. We experimented with this baseline approach using the edit distance. We implemented edit distance using PL/SQL and ran the baseline approach using a database of 60,000 strings from an internation movie database. The experiment was conducted in Oracle10g running on Redhat Enterprise Linux 3.0 Advanced Server. It took almost two days to accomplish on a dual 3Ghz Xeon Server. Therefore, faster algorithms are necessary to conduct string join.

In this chapter, we present a novel three step approach: first, we map the database of strings into points in an Euclidean space; then, we use approximate matrix multiplication to identify close pairs of points, whose pre-images are candidates for "similar" strings pairs; finally, these candidates are evaluated using the string distance to determine whether they are actually similar. We implement this method using Oracle PL/SQL for both efficiency and robustness. In this chapter, we use the popular edit distance. But there is nothing to restrict our methods to be applied to other metrics. To summarize, our contributions reported in this chapter are:

- A novel three step approach to detect approximately matching strings. Our method is implemented using database languages for both efficiency and robustness.
- Evaluation of our method using real world data sets. Experiment shows that our method is both efficient and accurate.
- A review the drawbacks of our methods. Then we extend them via string distance with block move for better performance.

The rest of the chapter is organized as follows. In section 2, we present background. We also present implementations of mapping algorithms using SQL. In section 3, we present the implementation of approximate matrix multiplication. In section 4, we describe the experiments, review the performance of our methods, and extend them to other more suitable string distances. In section 5, we review the most closely related work. Finally, in section 6, we draw conclusions.

5.2 Background

In this section, we introduce the popular metric mapping methods, namely, FastMap [33] and SparseMap[55]. We first list some relations (tables) used in our implementation. For simplicity, we omit the subscripts. For example, we use R to represent two string databases R_i, $i = 1, 2$.

- R(tid, doc): This relation holds the database of strings, which can be populated using SQL*Loader from a text file. Each tuple (tid, doc) holds a string doc with a unique tid.
- R_M(tid, cid, coord): A tuple (tid, cid, coord) indicates the coordinate of a string with tid in the dimension cid.
- Dists (t1, t2, dist): Dists is a temporary table, which means that its data is deleted after each iteration. A tuple (t1, t2, dists, dim) indicates the distance of string t1 and t2 computed so far.
- Sk(tid, doc): Sk is a temporary table that consists of a subset of R. Sk is used as a reference set in SparseMap. Its data is deleted after each iteration.
- Q(t1, t2): Q is a temporary table whose data is deleted after each iteration. A tuple (t1, t2) indicates the string t2 whose distance computed so far is the minimal to string t1.

The database of strings are mapped to an Euclidean space using one of these mapping methods. For any finite metric space (S, d), we can usually find a mapping F that maps the $|S|$ objects into a vector space of sufficient high dimensionality k, such that the distances between the objects in S are approximately preserved when using a proper distance function d' in the k-dimensional space. The mapping F must be constructed using substantially less then $O(|S|^2)$ distance computations. (Otherwise, the mapping method is no better than the baseline pairwise comparison approach.)

There are several metrics to measure the quality of a mapping F:

- Distortion: distortion is defined as

$$min\{c_1c_2 : \frac{d(s,t)}{c_1} \leq d'(F(s),F(t)) \leq c_2 \cdot d(s,t)\},$$

  for all $(s,t) \in S \times S$. The smaller the distortion, the better quality F has.

- Stress: stress is defined as

$$\frac{\sum(d'(F(s),F(t)) - d(s,t))^2}{\sum d(s,t)^2}.$$

  The summation is over all $(s,t) \in S \times S$. Again, the smaller stress, the better quality F has.

In the context of approximate string join, the above metrics of mapping quality are not informative since we care about the percentage of pairs we found and the number of pairs of candidates. We modify the definition of "cost" in [68] and use it as the metric to measure the quality of mappings later in this section. Next, we introduce the most popular mappings: FastMap and SparseMap. Readers are encouraged to read the paper [52] by Hjaltason et al., where the embedding methods are discussed in detail.

5.2.1 FastMap

FastMap [33] iterates to find two strings (pivot objects), which should be the farthest pair of strings. A line is drawn between them to form a coordinate axis and the coordinate value on this axis for each string s is determined by projecting s to this axis. Given a dimension k, FastMap iterates k times to form k orthogonal axis and maps each string into a point in the k-dimensional Euclidean space.

The following code shows a snapshot of the implementation of FastMap in PL/SQL. The getDistanceAll function, which is omitted here, computes the distance from a string to all other strings and returns the *tid* of the string with largest

distance. The second parameter indicates whether these distances should be stored in the temporary table Dists.

```
procedure fastmap (k number) is
a number; b number; dist number;
begin
    for i in 1..k loop
       choosePivot(a,b);
       select dist into dist from dists where t1=a and t2=b;
       insert into r_m
          select ta.t2, i,
          (ta.dist * ta.dist + dist*dist - tb.dist * tb.dist)/(2 * dist)
              from (select t2, dist from dists where t1=a) ta,
                   (select t2, dist where t1=b) tb
            where ta.t2=tb.t2;
    end loop;
end;
/

procedure choosePivot (a out number, b out number) is

begin
    a := 1;
    for i in 1..4 loop
       b := getDistanceAll (a, 0);
       a := getDistanceAll (b, 0);
    end loop;

    b := getDistanceAll(a, 1);
a := getDistanceAll(b, 1);
end;
/
```

Ideally, the choosePivot procedure should get the farthest pair of strings. Unfortunately, determining such a pair of strings would require pairwise comparison, which is computationally prohibitive. Faloutsos [33] proposed a heuristic to determine an approximately farthest string pair. The heuristic first chooses an arbitrary string a and then finds the string b with largest distance. Then it finds the farthest string for b. The process is iterated several times. Our experiments show that there is

little difference between (1) choosing the first string from the string database and iterating 3 times and (2) randomly choosing a string and iterating many (> 3) times. In our implementation, we iterate 5 times to approximately determine the pair of pivot strings. Therefore, FastMap requires $10kN$ edit distance computations, where k is the dimension of the embedding space and N is the number of strings in the database.

5.2.2 SparseMap

SparseMap [55] is based on the well-known Lipschitz embeddings [10], with two heuristics to reduce the construction time and dimensionality of Lipschitz embeddings. The first heuristic is a loop interchange which aims to reduce the number of string distance computations. The second heuristic, greedy re-sampling, reduces the dimensionality of the resulting embedding space.

The following code shows the implementation of the first heuristic of SparseMap.

```
procedure sparsemap (k number) is
begin
    for i in 1..k loop
       setSk(i);  mapdimension(i);
    end loop;
    end;
/

procedure setSk (k number) is
   rows number; size number; qry varchar2(200);
begin
    select count(*) into rows from r;
    size := power (2, floor(log(2,rows));
    execute immediate 'drop table sk;
    qry := 'create global temporary sk as select * from r sample ('
           || '100*size/rows ||);
    execute immediate qry;
end;
/

procedure mapdimension (k number) is
```

```
begin
    if k=1 then
       execute immediate 'insert into r_m select r.tid, 1, '||
           ' min(ed(r.doc, sk.doc))'||
           ' from r, sk where r.tid=sk.tid group y r.tid';
       return;
    end if;
    execute immediate 'insert into dists select r1.tid t1, r2.tid t2,'||
    ' sqrt(sum((r1.coord - r2.coord) * (r1.coord - r2.coord))) dist '||
    ' from r_m r1, r_m r2, sk' ||
    ' where r2.tid = sk.tid and r1.cid = r2.cid group by r1.tid, r2.tid;
    execute immediate
    'insert into r_m select r1.tid, k, ed(r1.doc, r2.doc)'
            ||' from r r1, r r2, q' ||
          ' where r1.tid = q.t1 and r2.tid = q.t2';
end;
/
```

For each dimension $1 \leq i \leq k$, a subset S_i of strings is generated, which serves as the reference set for the $i-th$ dimension. (For an explanation of the concept of reference set, please refer to [10].) The size of S_i is determined from the size of the database and the i^{th} dimension. For the first dimension, each string's coordinate is obtained from it's distance to S_1, where $d(s, S_i) = min\{d(s,t) : t \in S_i\}$. In the above code, ed is the edit distance. For higher dimension $j > 1$, the coordinate of a string s is determined as follows: for each string t in S_j, we compute the approximate distance $\delta(s,t) = \sqrt{\sum_{l=1}^{j-1}(s_l - t_l)^2}$, where s_l is the l^{th} coordinate of s and similar to t. Then we find the string t with smallest δ distance to s and compute their true distance, which becomes the j^{th} coordinate of s.

From the implementation, we see that it requires kN distance evaluations to map the N strings to k dimensional Euclidean distance. We found that SparseMap runs much faster than FastMap since SparseMap accesses the database with fewer disk I/Os.

5.2.3 Quality of Mappings

To measure the quality of a mapping in the context of string join, we define the metric "cost" as follows:

1. Run the mapping algorithm (either FastMap or SparseMap) against R_1 and R_2, for a dimension k.
2. Randomly select a small percentage (say, 2%) of strings from R_1 and R_2. Used the baseline approach to find all similar string pairs within a certain threshold δ.
3. Compute their Euclidean distances of these pairs and put them into equal depth buckets, where the number of buckets is $\frac{1}{\varepsilon}$. Let δ' be the left boundary of the last bucket.
4. Define cost as follows:

$$cost(d, \varepsilon) = \frac{|\{(s,t) : d'(F(s), F(t) < \delta'\}|}{|\{(s,t) : d(s,t) < \delta\}|} \tag{5.1}$$

The definition of "cost" is very similar to that in [68]. Note that, when we allow a small percentage ε (we call it the confidence level) of missing true positives (i.e., recall=$1 - \varepsilon$), the cost is reduced significantly. The reason we define cost as above is, that the mapping algorithms can map similar strings to arbitrarily far away points in the embedding space. To cut off these extreme cases, the mapping preserves distances pretty well. The left part of Figure 5.1 confirms that cost drops dramatically as ε increases. The right part of Figure 5.1 (roughly) suggests that cost decrease as dimensionality increases. The experiment shown in the left part of Figure 5.1 is conducted with the following constraints: Both R_1 and R_2 have 30,000 strings, we use the SparseMap and dimension $k = 40$. The experiment shown in the right part of Figure 5.1 is conducted with the following constraints: Both R_1 and R_2 have 30,000 strings, we use the SparseMap, and $\varepsilon = 0.05$.

To improve the quality of mappings, [55] proposed a heuristic called greedy re-sampling. We modify it slightly such that it works for the approximate string join case. After we have selected k reference sets, each single reference set is compared

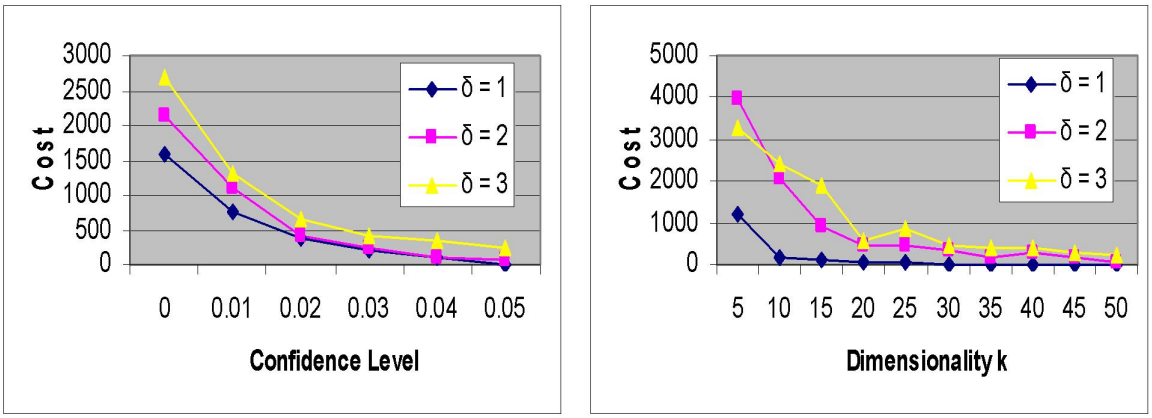

Figure 5.1. Cost versus confidence level ε and cost versus dimensionality k

in terms of the "cost(1, 0)" (note that cost is the function defined in Equation 5.1) of the embedding using only that reference set. Once the best reference set S_1 with smallest cost(1,0) is selected, we can pick the second reference set S_2 as the one which produces the best "cost(2,0)" when combined with S_1. The process can be repeated until we have reordered all k reference sets by decreasing order by quality. Experiments show that greedy re-sampling reduces the "cost", as shown in Figure 5.2. The experiment is conducted with the following conditions: We run the SparseMap with and without greedy re-sampling for dimensions 10 to 50. Both R_1 and R_2 have 30,000 strings and $\delta = 3$.

5.3 Similarity Join in Euclidean Space and Approximate Matrix Multiplication

In the previous section, we have mapped the database of strings to an Euclidean space and determined a new threshold δ', which is the left boundary of the last bucket. The second step tries to find (almost) all candidate pairs of close points in the Euclidean space. In the third step, we check each candidate pair by evaluating their edit distance to see whether they form a similar string pair.

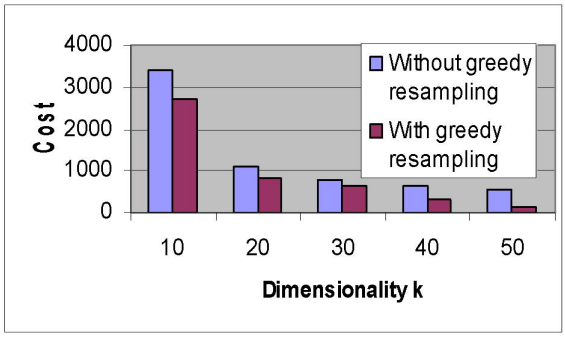

Figure 5.2. Cost reduced by greedy resampling

Note that for any two points x and y in R^k, their Euclidean distance can be computed $\|x-y\|^2 = \|x\|^2 + \|y\|^2 - 2 < x, y >$, where $< \cdot, \cdot >$ is the inner product. Therefore, to identify $\{(x, y) : \|x - y\| < \delta'\}$ is equivalent to identify

$$\{(x, y) :< x, y >> \frac{\|x\|^2 + \|y\|^2 - \delta'^2}{2}\}. \tag{5.2}$$

Let A be the matrix formed by the images of R_1 and B be the matrix formed by the images of R_2. More precisely, every row of A is obtained by mapping a string in R_1 and similar to B. To compute $< x, y >$ for all pairs of x and y is equivalent to compute AB^t where B^t is the transpose of B. To analyze the complexity of the similarity join in the Euclidean space, without lose of generality, we assume that the sizes of both R_1 and R_2 are N. Then both A and B are of size $N \times k$. Therefore, the baseline approach to identify point pair set 5.2 runs in $O(kN^2)$ time since all $\|x\|^2$ and $\|y\|^2$ can be pre-computed in $O(kN)$ time. The following SQL code snippet shows a pure SQL implementation of the baseline approach to compute AB^t and check the original string distance to see whether the candidate pair is actually a similar string pair.

```
create table noms1 as
  select tid, sum(coord * coord) nom2 from r_m1 group by tid;
create table noms2 as
```

```
  select tid, sum(coord * coord) nom2 from r_m1 group by tid;

Select r1.tid t1, r2.tid t2
    from r_m1 r1, r_m2 r2, noms1 n1, nroms2 n2
    where r1.cid=r2.cid and n1.tid=r1.tid and n2.tid=r2.tid
    group by r1.tid, r2.tid
    having sum(r1.cooord * r2.coord)
           > sum((n1.nom2 + n2.nom2 - d * d)/
/

select r1.tid t1, r2.tid t2 from r r1, r r2
  where (r1.tid, r2.tid) in (
     select rm1.tid t1, rm2.tid t2
       from r_m1 rm1, r_m2 rm2, noms1 n1, norms2 n2
       where rm1.cid = rm2.cid and n1.tid = rm1.tid
         and n2.tid = rm2.tid
       group by rm1.tid, rm2.tid
       having sum(rm1.coord * rm2.coord)
              > sum((n1.norm2+n2.om2 - d * d)/2
          and ed(r1.doc, r2.doc) < d
/
```

5.3.1 Approximate Matrix Multiplication

The task here is to approximately calcuate AB^t. As we are aware, there exist two algorithms for approximate matrix multiplication. In [21], Cohen et al. propose a random-sampling based algorithm that identifies high-valued entries of nonnegative matrix products, without computation of the product. In [30], Drineas et al. proposed a randomized algorithm which randomly picks s columns of A to form an $N \times s$ matrix S and the corresponding columns of B to form an $N \times s$ matrix R. After scaling the columns of both S and R, we let $P = SR^t$, which is an approximation of AB^t.

We denote by A_j the j^{th} column of A as a column vector, similar to B. We denote by A^i the i^{th} row of A as a row vector, similar to B. Let $\|A_j\|$ be the Euclidean norm of the column vector A_j. let $|A_j|$ be the l_1 norm of the column vector A_j, i.e., $|A_j| = \sum_{i=1}^{N} a_{ij}$. Note that all entries in A (produced by the mapping) are

non-negative.

5.3.2 Implementing Fast Monte-Carlo in SQL

Here is the fast Monte-Carlo algorithm from [30] with some modification.

- for i=1 to k, we choose $p_i = \frac{\|A_i\|\|B_i\|}{\sum_{l=1}^{k} \|A_i\|\|B_i\|}$. Note that $\sum p_i = 1$.
- for t=1 to s independently
  - pick $i_t \in \{1...k\}$ at random with $prob(i_t = j) = p_j, j = 1...k$
  - include $A_{i_t}/\sqrt{sp_{i_t}}$ as a column of S and $B_{i_t}/\sqrt{sp_{i_t}}$ as a column of R
- return SR^t as the approximation of AB^t.

It can be proved that the expected value of the ij^{th} entry of SR^t is equal to that of AB^t.

We implement this sampling algorithm in PL/SQL using the DBMS_RANDOM package, which is used to generate random numbers in an Oracle database. Intuitively, after s random trials, each column A_j of A would have been picked $round(s \cdot p_j)$ times on average. We show the deterministic version of the sampling algorithm using SQL in Figure 5.3. Our experiments shows that the deterministic version works equally well as the randomized version and runs faster.

As we see from the figure, two new relations R_M11 and R_M22 are created. Replace R_M1 with R_M11 and R_M2 with R_M22 in the above code snippet, we obtain an approximate string join.

5.3.3 Implementing Cohen's Sampling Algorithm in SQL

Cohen's sampling algorithm tries to find out all large valued entries in a matrix before doing the actual product. For each column j, we sample a row vectorA^i with probability $\frac{a_{ij}}{|A_j|}$ and we performance s Bernoulli trials for each row. We can

```
create table norm1 as select cid, sqrt(sum(coord * coord)) nom from r_m1 group by cid;
create table norm2 as select cid, sqrt(sum(coord * coord)) nom from r_m2 group by cid;
create table total as select sum(n1.nrom * n2.norm) t from norm n1, norm2 n2
  where n1.cid = n2.cid;
create table p as select n1.cid, n1.norm * n2.norm/t prob from norm1 n1, norm2 n2, total
 where n1.cid = n2.cid group by n1.cid, n2.cid;
create table r_m11 as select tid, p.cid, coord * sqrt(s * prob) coord from r_m1, p
 where r_m1.cid = p.cid and round(s * prob, 0) > 0;
create table r_m22 as select tid, p.cid, coord * sqrt(s * prob) coord from r_m2, p
 where r_m2.cid = p.cid and round(s * prob, 0) > 0;
```

Figure 5.3. SQL implementation of a deterministic version of the fast Monte-Carlo algorithm

```
create table norm12 as select cid, sum(coord) n1 from r_m2 group by c1;
create table r_m2s as select r_m2.tid, r_m2.cid, round(s*r_m2.coord/norm12.n1, 0) coord
  from r_m2, norm12
  where r_m2.cid = norm12.cid and round(s*r_m2.coord/norm12.n1,) > 0;

select r1.tid t1, r2.tid t2 from r r1, r r2 where (r1.tid, r2.tid) in (
  select rm1.tid t1 r_m2.tid t2
    from r_m1 rm1, r_m2s rm2, norm1 n1, norm2 n2
    where rm1.cid = rm2.cid and n1.tid = rm1.tid and n2.tid = rm2.tid and rm1.cid = n12.cid
    group by rm1.tid, rm2.tid
    having sum(rm1.coord * rm2.coord * n1.n1) > sum((n1.norm2 + n2.norm2 - d' * d')/2))
      and ed(r1.ddoc, r2.doc) < d;
```

Figure 5.4. SQL implementation of a deterministic version of Cohen's sampling algorithm

implement this sampling algorithm using PL/SQL. Again, a deterministic version of the sampling algorithm can be implemented using SQL. We sample on the relation R_M2 as shown in Figure 5.4 .

It's interesting to note that Cohen's sampling algorithm is not symmetric, which is different from the fast monte-carlo algorithm. Figure 5.4 shows a sampling performed on R_M2. We can also perform sampling on R_M1 and combine the sampling to form other variations of sampling method. [78] gives detailed discussions of these variations.

5.4 Experiments and Discussions

Our method is a three step approach: we first map the databases of strings to points in an Euclidean space; then we conduct approximate similarity join in the Euclidean space using approximate matrix multiplication; finally, the candidates are evaluated using original string distance to filter false positives. We implemented our approach in a commercial RDBMS. The rest of this section is organized as follows: In Section 5.4.1, we describe the datasets and techniques used for our experiments. In Section 5.4.2, we report the results and perform discussions.

5.4.1 Settings

We implemented the algorithms described in Section 5.2 and Section 5.3 in Oracle10g, running on a dual CPU Redhat Enterprise Linux 3.0 Advanced Server, with 2×3GHz Intel Xeon CPUs and with 2Gb of RAM. The mapping algorithms, FastMap and SparseMap, are implemented using PL/SQL. The similarity join in an Euclidean space is implemented using only SQL, with a string edit distance implementation using PL/SQL.

Datasets:We collected a sample of 60,000 movie star names and movie names without duplicates from the international movie database. (For any datasets that have duplicates, a sorting method can be used to remove duplicates.) Then we split the sample randomly into two datasets R_1 and R_2, each consists of 30,000 strings. For both R_1 and R_2, their average string length is around 14. The distribution of lengths of strings is close to a normal distribution.

Measurements: Let A_δ be the result of an approximate string join and let B_δ be the result of a baseline string join. The notions of precision and recall are defined as follows:

$$precision(\delta) = \frac{|A_\delta \cap B_\delta|}{|A_\delta|}$$

$$recall(\delta) = \frac{|A_\delta \cap B_\delta|}{|B_\delta|}$$

Precision captures the proportion of string pairs in A_δ that are actual similar string pairs in B_δ. Recall captures the proportion of correct similar string pairs in B_δ found in A_δ. A 100% precision indicates that all string pairs found are actually similar strings. However, it does not mean all similar string pairs are found in the approximate string join. In the context of approximate string join, recall is more important then precision, since false positives can be filtered after the join (as long as precision is not too small, otherwise, the algorithm is no better than a baseline pairwise approach).

Techniques: We compare FastMap and SpaseMap for mapping strings to points. We compare fast Monte-Carlo and Cohen's sampling methods for approximate matrix multiplication. In addition, we also compare our approach with the approach in [46] where three filtering mechanisms, namely, length filter, count filter and position filter, are implemented using SQL statements.

5.4.2 Results and Discussions

Comparing mapping algorithms: We compare the following mapping algorithms:

- FastMap
- SparseMap without greedy resampling
- SparseMap with greedy resampling

We first compare their running time (seconds) with respect to the dimensionality k and data size, as shown in Figure 5.5. Note that the left figure is a log-log plot, and the slopes of the lines are close to 2; therefore, the mapping execution time is $O(k^2)$. From the right figure, we see that the mapping execution time is proportional to

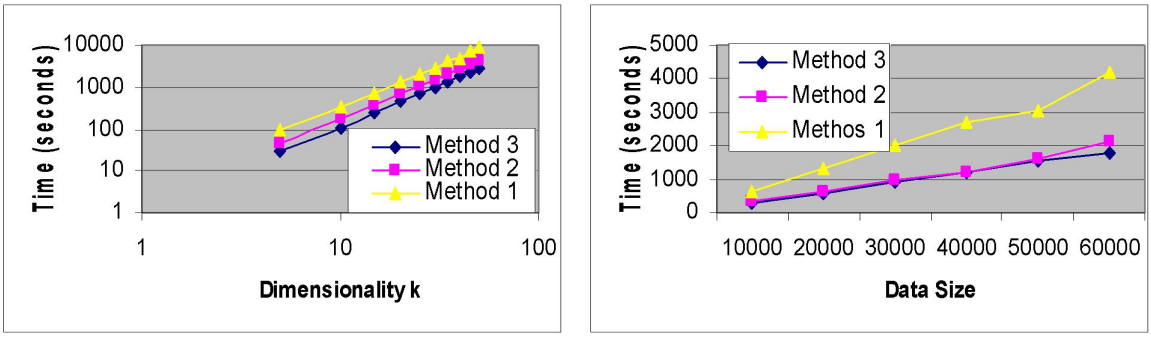

Figure 5.5. Execution time of mappings versus dimensionality k and data size

data size. From Figure 5.5, SparseMap runs faster than FastMap for fixed datasize and dimensionality, even SparseMap with greedy resampling is faster then FastMap.

Table 5.1 shows the new threshold δ' after each mapping method, where dimensionality $k = 45$, confidence $\varepsilon = 0.05$, and datasize is 30,000. Method 1,2,3 denotes FastMap, SparseMap with greedy resampling and SparseMap without greedy resampling respectively. We see that SparseMap maps the points farther away than FastMap.

Table 5.1

NEW THRESHOLDS FOR DIFFERENT MAPPING METHODS

Method	$\delta = 1$	$\delta = 2$	$\delta = 3$
1	3.45	5.98	9.32
2	4.33	7.99	12.23
3	5.32	9.35	15.48

Figure 5.6 shows the trends of cost versus dimensionality for each mapping method. We see that FastMap has the best cost, SparseMap without greedy resampling has the worst cost and SparseMap with greedy resampling has almost the

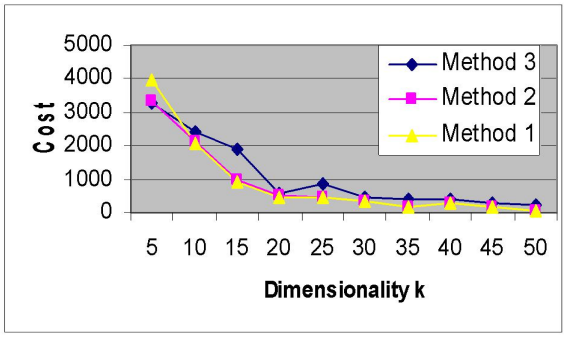

Figure 5.6. Cost versus dimensionality k for each mapping method

same cost as FastMap.

Comparing join algorithms: The similarity join in Euclidean space is realized using approximate matrix multiplication. In previous sections, we implemented both algorithms using pure SQL. We first compare the execution time for the fast Monte-Carlo and Cohen's sampling methods against the number of Bernoulli trials s. Then we compare the recall of both methods against the number of Bernoulli trials s. Figure 5.7 shows that with the same number of Bernoulli trials s, fast Monte-Carlo method runs slightly faster. Both methods achieve approximately the same recall. We see that as s increases, both execution time and recall increase.

Comparing with other approaches: We compare our approach with that in [46]. We use the SparseMap with greedy resampling method to map the two string databases R_1 and R_2 to R^{40}, the 40-dimensional Euclidean space. Then we use the fast Monte-Carlo method to find nearly all pair of close points and finally, we do a post-join filtering. We found 91.2% of strings pairs in less than 1 hour, while the method in [46] takes 12 hours.

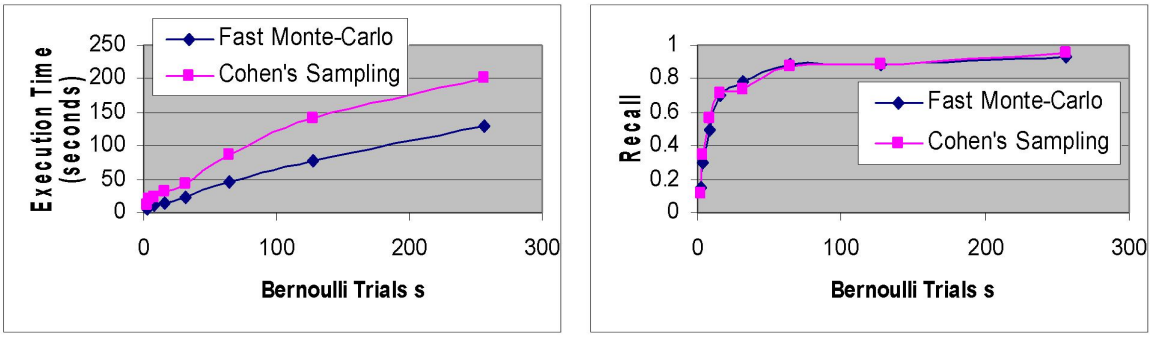

Figure 5.7. Comparing approximate matrix multiplication methods

5.4.3 Limitations of the Mapping Approach

A low distortion embedding of edit distance into normed space is very desirable. Unfortuately, so far essentially nothing is known about the embeddability of the edit distance into a normed space, except for generic embeddability results for metric space, which do not provide interesting bounds. The performance of the mapping algorithm may degrade if we cannot control the quality of mapping, since the post-join process will take a long time if two many candidates are found.

In [24], the edit distance is modified by allowing the moving any contiguous block of characters as a single operation, then the resulting block-edit metric can be embedded into l_1 with distortion $O(logd \cdot log^* d)$, where d is the longest length of the strings. In some cases, the block-edit metric may be more reasonable: for example, the two strings "University of Notre Dame" and "Notre Dame University" have large edit distance (20), while they have a small block-edit metric distance. If we use the block-edit metric, we can conduct the approximate string join in the l_1 space. The embedding space l_1 in [24] has very high dimensionality, therefore, some popular similarity join methods do not work. Approximate matrix multiplication method does not work for l_1 space either.

5.5 Related Work

Data cleansing has become an important field in recent years. Hernandez and Stolfo [51] studied the approximate duplicates detection problem using the sorted-neighborhood method. In their method, a key is generated and the database is sorted using this key. Then, they use the sliding window approach to detect approximate duplicates. Ananthakrishna [5] proposed an algorithm for eliminating duplicates in dimensional tables in a data warehouse. Chaudhuri [19] proposed a new similarity function and used this function to develop a online fuzzy match algorithm.

Li [68] proposed a two step approach for approximate string join. A database of strings is first mapped to an Euclidean space using StringMap, a variation of FastMap and then used a similarity join algorithm to detect candidates. The similarity join algorithm does not work in high dimensional space because of the "curse of dimensionality". Gravano [78] proposed a sampling method for approximate text join using the vector cosine metric. Cohen [22] used a clustering method to create clusters for potential duplicates. The work in [68] and [78] are closest to our work.

5.6 Summary

In this chapter, we designed a novel three step approach data cleansing algorithm based on approximate strings joins. We first map each string database to an Euclidean space and then conduct similarity join in the Euclidean space using approximate string join. Experiments show that this approach is both accurate and efficient. We also point out the limitations of the mapping approach: since the quality of the mapping can not be guaranteed, in order to find almost all pair of similar strings, we might need to check a huge number of string pairs, which may degrade the performance of our algorithm.

CHAPTER 6

A CASE STUDY: THE NOM SIMULATION PROJECT

We describe the agent-based stochastic simulation models of NOM transformations and its implementation using Java/Swarm/Repast. The simulation models are deployed to the Web for service. We design a portal using the best practice Model-View-Controller based architecture to facilitate remote invocation of simulations, analysis and visualization of the simulation datasets. The NOM simulation models may be useful in many areas including chemistry, geology, microbial ecology and environmental science. [1]

6.1 Introduction

Natural Organic Matter (NOM) is a complex mixture of compounds formed as a result of the breakdown of animal and plant material in the environment. NOM is ubiquitous in terrestrial, aquatic, and marine ecosystems, playing a crucial role in such important processes as the evolution and fertility of soils; the mobility and transport of pollutants such as trace metals, radionuclides and hydrophobic organic compounds; the availability of nutrients to micro-organisms and plant communities; the growth and dissolution of minerals; and the global biogeochemical cycling of the elements [3].

[1] Part of this chapter appeared in IEEE Computing in Science and Engineering [61]

NOM has a significant impact on all aspects of drinking water treatment. NOM is responsible for the majority of the coagulant demand. Therefore water with high dissolved organic carbon levels usually has a high coagulant requirement and consequent high treatment costs. NOM can cause major problems in the treatment of water as it reacts with chlorine to form disinfection by-products. Many of the disinfection by-products (DBPs) formed by the reaction of NOM with disinfectants, are reported to be toxic and carcinogenic to humans if ingested over an extended period of time. The removal of NOM and hence reduction in DBPs is a major goal in the treatment of any water source.

The importance of NOM attracts numerous researchers, including chemists, geologists and even computer scientists. Despite decades of research, we still know relatively little about the structure, chemical composition, and chemical properties such as molecular weight, functional group concentrations, structure, composition, and reactivity [87, 14, 71].

During the last 50 years, some research has been done on the functional behavior of NOM molecules, and different models are proposed to handle NOM research, including ODE (Ordinary Differential Equation), PDE (Partial Differential Equation), etc (see Section 6.2). But the diversity of the compounds present in NOM leads to the difficulty of describing the mixture adequately and makes models computationally expensive.

In this chapter, we describe a new agent-based stochastic modeling approach to simulate the behavior of NOM. The simulation was implemented using Java, the Swarm library [115] and RePast [99]. To make the simulation accessible by scientists around the world, we employ the J2EE (Java 2 Enterprise Edition) and RDBMS (Relational DataBase Management System) technologies.

The rest of the chapter is organized as follows. Section 2 identifies some previous

simulation models and points out their limitations. Section 3 presents our new agent-based stochastic model of NOM, performance discussions and model verification and validation. Section 4 introduces the information technologies involved in our work. Section 5 describes the project portal. Section 6 provides detailed checkpoint and restart processes for the NOM simulation. Section 7 describes our experience on building the NOM data warehouse. Finally, section 8 summarizes this chapter.

6.2 Background

Despite decades of research, we still know relatively little about the structure, chemical composition and reactivity of NOM primarily because it is a complex mixture of molecules with different physico-chemical properties such as molecular weight, functional group concentration, structure, composition and reactivity. The evolution of NOM from its biological precursor compounds is both an interesting biogeochemical problem and an important aspect of predictive environmental modeling. On one hand, carbon cycling models based on average properties of various organic carbon pools are too simplistic to represent the heterogeneous structure of NOM and its complex behavior in the environment. On the other hand, molecular models employing connectivity map or electron density are too computational expensive to be useful for large-scale environmental simulations.

With awareness of the drawbacks of the previous models, we propose a middle path: development of an innovative stochastic model that will allow, for the first time, forward modeling of the evolution of NOM structure and properties. This stochastic model of NOM evolution represents individual molecules as discrete objects of specified elemental and functional group composition, size and reactivity. Temporal evolution of NOM from biological precursor compounds such as lignin, polysaccharides and proteins is simulated using Monte Carlo algorithms in which specific

probabilities are assigned to particular transformations. These algorithms employ newly-developed pseudo-random number generators with long periods and robust statistical properties provided by the Swarm library. Both physico-chemical and biochemical effects are incorporated probabilistically. The reactivity of the resulting NOM assemblage over time is predicted based on the distributions of molecular properties.

This stochastic approach has several advantages: it is much less computationally intensive than molecular modeling or explicit kinetic simulation of hundreds of compounds, it can readily be adapted to a variety of time scales and processes, and it intrinsically handles NOM structural and functional heterogeneity.

6.3 Agent-based Stochastic Model

In this section, we present the stochastic simulation model using agent-based technology. This stochastic model of NOM represents individual molecules as discrete objects of specified elemental and functional group composition, size and reactivity. Temporal evolution of NOM from biological precursor compounds such as lignin, polysaccharides and proteins is simulated in which specific probabilities are assigned to particular transformations. The reactivity of the resulting NOM assemblage over time can be predicted based on the distributions of molecular properties.

6.3.1 Modeling of NOM Evolution

Since representing each molecule as a detailed chemical structure is prohibitively computationally expensive, we design a data structure which is much simpler to work with but which encapsulates much of what we know and when we wish to know about NOM. The data structure includes the following:

- The elemental formula, i.e., the number of C, H, O, N, S and P atoms in the molecule.

- A record of the molecular "origin", i.e., the type and size of precursor molecule and the time at which it was entered into the simulation. This allows the calculation of separate "turnover time" and apparent ages for individual molecules and fractions.
- Functional group counts: carboxylic acid, alcohol, ester, keton, aldehyde, thiol, sulfato, amine and peptide groups are counted in each molecule.

The NOM simulation is implemented in a discrete 2D space with discrete time. The simulated space is a rectangular lattice. Each molecule can occupy at most one cell, and each cell can host one or more molecules. The molecules in the simulation are intended to represent a sample from a large population in the system under study. During execution of the simulation, each molecule may move to another cell or stay in a fixed cell according to predefined simple rules that describe physical processes. In chemical reactions, one molecule could split; two or more molecules could combine and occupy just one cell.

There are 10 types of chemical reactions represented in the simulation system: ester condensation, ester hydrolysis, amine hydrolysis, microbe uptake, dehydration, strong C=C oxidation, mild C=C oxidation, alcohol (C-O-H) oxidation, aldehyde C=O oxidation, and decarboxylation. Each molecule has a probability for each type of chemical reaction. The calculation of the probability is based on the structure of the molecule, and the environment in which the molecule resides. These environmental variables include the length of the time step, microbe density, fungal density, pH value, temperature, pKw (the equilibrium constant for the autolysis of water, which is very close to 14.0), oxygen density, light density, etc. After each chemical reaction, the probabilities of these reaction types are re-calculated. The reaction probabilities are stored in an array associated with each molecule.

In each time step, for each molecule, a random number is generated which is used to determine whether a chemical reaction will occur, and if one occurs, which reaction type. In the simulation, the sum of all the reaction probabilities is controlled to be less than 1 percent. The interval $[0, 1]$, is partitioned into 11 subintervals. The length of the first interval is equal to the probability of the first reaction type; the length of the second interval is equal to the probability of the second reaction type, etc. The length of the last interval is the probability in which no reaction will occur. The generated random number from the interval [0,1) will reside in one of these intervals, and it will decide which chemical reaction will occur, if any at all.

If the chosen chemical reaction type is a second order reaction (the probability of higher (> 2) order reactions is so small that we can safely ignore them), i.e., there will be two molecules involved in the reaction, the second molecule will be chosen from one of its nearest neighbors who are not yet involved in a chemical reaction. After the reaction takes place, the probability tables for involved molecules are updated at the end of the time step and the new probability tables are assigned to newly produced molecules.

The NOM simulation is implemented using Java/Swarm/RePast, the Oracle RDBMS, JDBC (Java Database Connectivity) and SQL*Loader.

6.3.2 Performance Discussion

Agent-based modeling (ABM) (also known as individual-based modeling (IBM) [50]), and equation-based modeling (EBM) are two approaches for modeling complex systems. ABMs and EBMs are significantly different with respect to which characteristics they focus on. ABMs focus on the characteristics of each individual and track them through time. EBMs, on the other hand, focus on the characteristics of the population, which are averaged and simulation changes in the averaged

population characteristics [94]. It involves a process for solving a set of differential equations. EBMs can describe already known global properties of a system, but often can neither explain the origin of those properties nor track the behavior of individual components. The objectives of ABMs are describing the heterogeneous aspects of individual agents and system, providing a mechanism for interactions between individual agents, and predicting phenomena at higher levels based on the actions of individual agents, in a bottom-up approach. The NOM complex system consists of a large number of heterogeneous molecules and microbes. Individual molecules can be transported through the soil medium via water flow, adsorb on the soil particle surfaces, and react with other molecules or microbes. The properties of each individual molecule can change over time. Also, new molecules are emerging (one molecule is split into two) and molecules disappear (reacting with microbes) in the system. It is hard to capture the global properties of the NOM system accurately with EBMs modeling approach. Cabaniss [12] presents several examples showing that the use of "average" values to represent the complexity of NOM is problematic. It can result in discrepancies when the model results are compared with the results from the laboratory studies. Therefore, ABMs is more suitable for modeling the NOM complex system that is composed of interacting individuals and exhibits a wide range of dynamic behavior. Agent-based modeling of molecules as objects is also being investigated in Cell Biology. For example, at Cambridge University, in the UK Shimizu et al. [90, 109, 108] create a stochastic simulator for chemical reactions among molecules. In their model, each molecule is represented as an independent software object, called agent in our model.

The agent based approach is more flexible and accurate to model the evolution of NOM. However, compare to the equation based modeling approach, it is much slower. The performance becomes an important issue for the agent based simulation.

Determining and understanding the factors that affect the performance of simulation from a software engineering perspective and identifying and eliminating the bottlenecks that limit scalability at the software development stage are necessary for achieving high performance. Several approaches for exploring the performance and scalability improvement of the simulation model are taken. These approaches include runtime optimization, database access, objects usage, and parallel and distributed computing. We discovered that by employing these technologies and implementing the distributed simulation model with MPJ [15], the performance can be improved by 25 times. The performance is expected to increase more while the number of agents in the system increases. The advantages (flexibility and accuracy) of using ABMs in the NOM system may overcome the performance concern.

6.3.3 Verification and Validation (V & V)

Verification and validation (V & V) are processes to increase confidence in simulations. Verification is about getting the "simulation right" while validation is about getting the "right simulation". Although neither process guarantees absolute confidence, we used numerous V & V techniques on the NOM simulation. We describe these techniques using an adapted version of Sargent's V & V process shown in Figure 6.1 [104]. The simulation process starts with an identification of research questions of interest. Through analysis and modeling, a conceptual NOM model is developed that includes the important features relevant to the research question. The conceptual model is based on theory and domain knowledge from environmental chemistry. This theoretical foundation includes 1) the heterogeneity of NOM molecules, 2) the important NOM iterations with mineral surfaces such as adsorption, hemi-micelle formations, acid or complexing dissolution, and reductive dissolution, 3) NOM interaction with pollutants, 4) relationships between NOM adsorption to

mineral surfaces and the molecular weight of the NOM molecules, and 5) probabilistic reaction kinetics based on elemental composition and the nature of functional groups in the molecules. The incorporation of such theory and domain knowledge provides us initial face validity, i.e., the logic of the conceptual model appears to domain experts to include appropriate mechanisms and properties of the research problem. Six scientists - two biologists, a chemist, a geomicrobiologist, and two soil scientists - evaluated the conceptual model for face validity. Once the conceptual model achieves its initial validation, coding of the agent-based simulation took place. At this step, verification methods such as code walk through, trace analysis, input-output testing, and boundary testing were used to verify the correctness of the simulation. To date, the validation of the simulation has included comparisons of simulation behavior with mathematical models and experimental results. For example, the simulation has been used to study the relationship between the adsorption of NOM molecules on mineral surfaces and the molecular weight of the molecules. This relationship has been empirically observed to have a log-normal distribution. The simulation has yielded a similar log-normal distribution as reported by Arthurs et al [7]. Additional such comparisons between theoretical, empirical and simulation predictions have been completed.

6.4 Simulation Technologies

In the simulation described in this chapter, multiple information technologies are integrated to investigate a new paradigm of scientific inquiry. Java is reaching performance parity with other languages (e.g., C/C++) and has begun to be used in scientific simulations [41] [119] [122]. With the release of J2SE 1.5, management and monitoring features can be inserted into the simulation programs to monitor the Java Virtual Machine and the underlying operating systems.

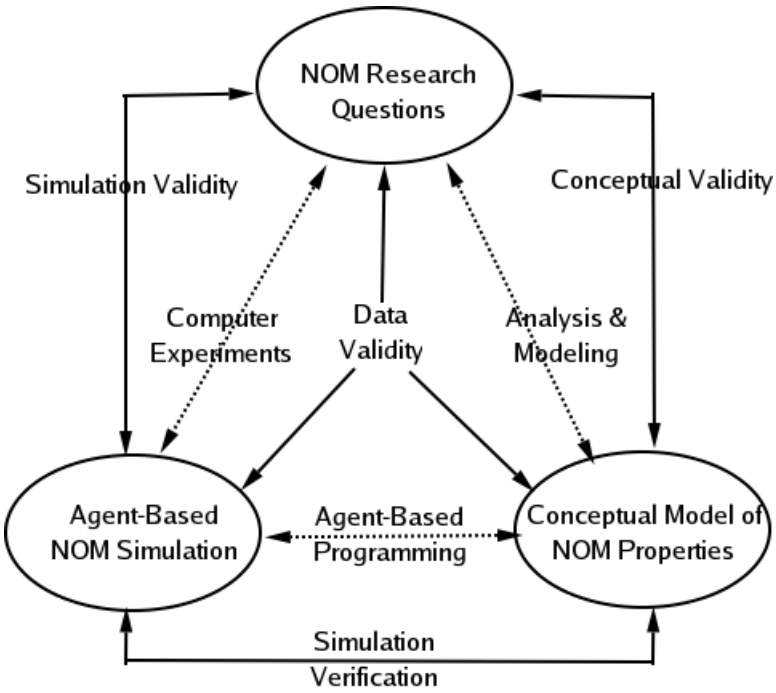

Figure 6.1. The verification processes of the agent-based stochastic model

Fox examined the paradigms of scientific study and introduced a fourth paradigm [37]. We used to speak of three approaches to science: experiment, theory and the computational approach. The fourth paradigm of scientific study uses IT technologies such as web, databases and data mining. We support the molecular simulation models using these IT technologies.

The NOM simulation system includes a portal which allows users to provide inputs and to obtain data analysis reports for their simulations through Web browsers.

The NOM is implemented using J2EE (Servlets, JavaServer Pages, JavaServer Faces and Enterprise Java Beans) with the best practice MVC (Model-View-Controller) architecture, running on Oracle9i Application Servers. Simulation data is stored into Oracle databases through Java Database Connection (JDBC), and is extracted, transported, transformed and loaded into a data warehouse using SQL*Loader and

PL/SQL procedures. Reports of the simulation results are generated using Java, SQL, PL/SQL, XSQL and JFreeChart.

In the following section, we briefly introduce these technologies.

- Java: Java technology is both a programming language and a platform. The Java platform has two components: (1) the Java Virtual Machine (JVM) and (2) the Java Application Programming Interface (Java API). In the J2SE 1.5 release, the Management and Monitoring APIs enable monitoring of the Java Virtual Machine and the underlying operating systems inside Java programs.

- Swarm/RePast: Swarm is a set of APIs developed at the Santa Fe Institute. From the documentation: "Swarm is a muti-agent software platform for the simulation of complex adaptive systems. In the Swarm system the basic unit of simulation is the swarm, a collection of agents executing a schedule of actions. Swarm supports hierarchical modeling approaches whereby agents can be composed of swarms of other agents in nested structures. Swarm provides object oriented libraries of reusable components for building models and analyzing, displaying and controlling experiments on those models." RePast is a software framework for creating agent based simulations using Java. It borrows much from the Swarm simulation toolkit and can properly be termed as "Swarm-like".

- Oracle RDBMS/Data Warehousing/Data Mining: The purpose of databases is to store and retrieve related information. A database server is used to solve the problems of information management. In general, it reliably manages a large amount of data in a multi-user environment so that many users can concurrently access the same data. A database server also prevents unauthorized access and provides solutions for failure recovery. In our work, we use the Oracle database servers to support data analysis, data warehousing and data mining. A data warehouse is a relational database that is designed for query and analysis. In addition to a relational database, a data warehouse environment includes an extraction, transportation, transformation and loading (ETL) solution, an online analytical processing (OLAP) engine, client analysis

tools and other applications such as Data Mining that manage the processing of discovering patterns and trends inside the data.

- JDBC: JDBC(Java Database Connectivity) is a standard Java interface for connecting from Java to relational databases such as Oracle. The JDBC standard was defined by Sun Microsystems, allowing individual providers to implement and extend the standard with their own JDBC drivers. JDBC is based on the X/Open SQL Call Level Interface and complies with the SQL92 Entry Level standard. In our work, we use Oracle JDBC Thin driver, which allows a direct connection to the database by providing an implementation of SQL*Net and TTC (the wire protocol used by Oracle Call Interface (OCI)) on top of Java sockets and runs over TCP/IP.

- J2EE: The Java 2 Enterprise Edition (J2EE) has historically been an architecture for building server-side deployments in the Java programming language. It can be used to build traditional web sites, software components, or packaged applications. We use the following J2EE technologies in our work: (1) Java Servlet, (2) JavaServer Pages (JSP), (3) JavaServer Faces (JSF), and (4) Enterprise JavaBeans (EJB). Java Servlets provide a component-based, platform-independent method for building Web-based applications. JSP technology is an extension of the servlet technology created to support authoring of HTML and XML pages. It makes it easier to combine fixed or static template data with dynamic content.

- JSF: JavaServer Faces (JSF) technology simplifies building user interfaces for JavaServer applications. JSF technology includes (1) a set of APIs for representing user interface (UI) components and managing their state, handling events and input validation, defining page navigation, etc. and (2) a JavaServer Pages (JSP) custom tag library for expressing a JavaServer Faces interface within a JSP page. Oracle Application Server provides a J2EE compliant platform to develop and deploy Internet applications, Web sites and portals. We use Oracle Application Server as the platform to deploy web applications which provide interfaces to the users and visualize browser-based data analysis results.

- XSQL: XSQL is a servlet tool that processes SQL queries and outputs the result set as XML. It is the combination of XML and SQL to provide a language and database independent means for storing SQL queries, clauses and query results.

- JFreeChart: JFreeChart is a free Java class library for generating charts, including pie charts, bar charts, line and area charts, scatter plots and bubble charts, time series charts, combination charts, etc. We are using JFreeChart to visualize data analysis results. The dynamic graphs created by JFreeChart are embedded in JSP pages.

6.5 The NOM Portal

The NOM portal is a web application adhering to the Model/View/Controller (MVC) design pattern. It is implemented using two existing J2EE application framework: Apache Struts and Oracle Application Development Framework(ADF). Both the Struts and ADF echo the pragmatic suggestions that Johnson details throughout his book [69]. The NOM portal has the basic architecture illustrated in Figure 6.2.

- The *model* layer represents the information needed by the application, it is implemented using a combination of EJB, JavaBeans and Business Components for Java (BC4J).

- The *controller* layer handles user input, interfaces with the model layer, and picks the presentation. It is implemented using Apache Struts [114]. The core of the Struts framework is a flexible control layer based on standard technologies such as Java Servlet, JavaBeans, ResourceBundles and XML, as well as various Jakarta Commons packages.

- The *view* layer presents the model data to the end-user. It is implemented using a combination of JSP, UIX (User Interface XML) and JSF.

The model layer consists of business services that expose application functionality and access to model data through a business service interface. These business

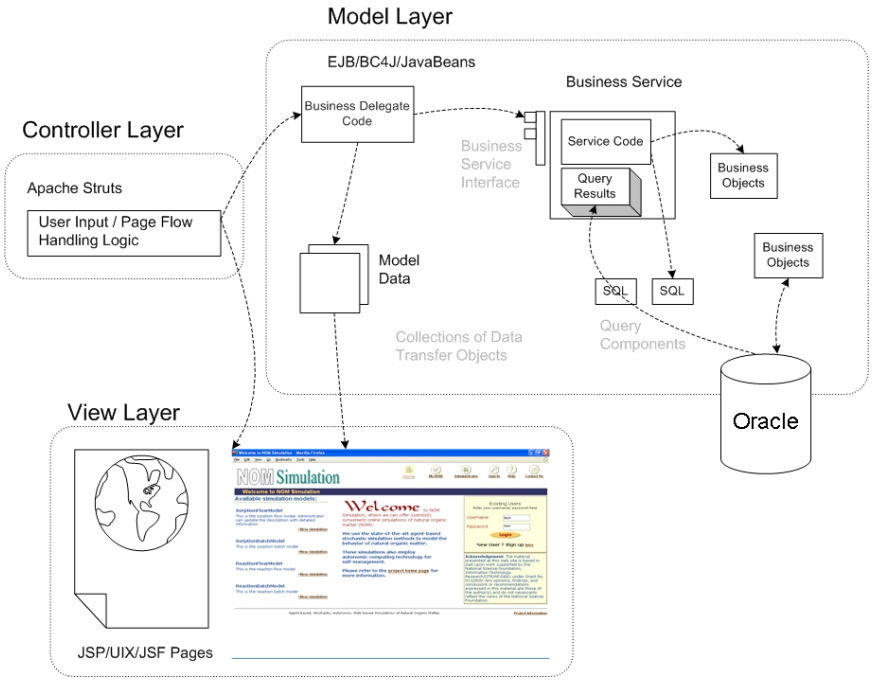

Figure 6.2. The MVC Model 2 NOM Portal Architecture

services, in turn, rely on query components to retrieve data and on business objects to validate and persist any new or modified data. Code implementing the business delegate design pattern abstracts the details of locating and using the business services. The architecture we use in our work to design the web application is the best practice "JSP Model 2" architecture. The number "2" is used because this MVC-based architecture is an evolution over the first generation of JSP-based approaches.

6.6 Checkpoint and Restart

From Chapter 2 and Chapter 3, we learned that

- The optimal checkpoint interval is independent of the restart time. However, the expected simulation completion time is exponentially dependent on the restart time and thus a fast failure detection yields faster simulation completion time.

- The optimal checkpoint interval and expected simulation completion time increase monotonely as checkpoint time increases.

In this section, we describe what data are to be checkpointed and how they are retrieved for restarting. Since the checkpoint time and restart time are very much related to the simulation completion time, we propose and compare different ways to do checkpoint and restart.

6.6.1 Checkpoint

Basically, everything required to restore the simulation state needs checkpointed. For the NOM simulation, checkpointed data include

- Molecule composition information.
- Reaction probabiilities for each reaction type.
- Molecule status, for example, whether adsorbed.
- Current and previous positions in the environment for each molecule.

The checkpoint data is eventually stored in the database. Different ways to store checkpoint data include JDBC and SQL*Loader. Experiments comparing the performance for each method show that SQL*Loader is a number of times faster than JDBC. Therefore, we use SQL*Loader to do checkpoint. The procedure is as follows:

1. Checkpoint data is first written to a text file on local drive of a simulation server.
2. The data in the text file is loaded into the database through SQL*Loader. A log file is generated when doing SQL*Loader. If the load fails, error messages are generated in the log file. The data is re-loaded if error message exists in the log file. This is done by the autonomic agent on the simulation server.

The checkpoint process takes about 5 seconds to accomplish in average.

6.6.2 Restart

When a failure occurs, the checkpoint data inside the database is retrieved for restarting the simulation. This retrieval is accomplished through JDBC. There are several ways to retrieve data through JDBC certainly, our experience tells us batched PreparedStatement outperformances its counterpart Statement. The JDBC driver makes two round trips to the database for each select statement. On the first roundtrip, it retrieves the metadata for the columns selected. On the second round trip, it retrieves the actual data selected. With this in mind, the performance for select statements can be improved by roughly 50% if we define the select statement by using Oracle's defineColumnType() method with an OracleStatement object. When we predefine a select statement, we provide the JDBC driver with the column metadata using the defineColumnType() method, obviating the need for the driver to make a round trip to the database for metadata information.

The average restart time is the summation of two periods:

- Period where system is not aware of the failure before detection.
- Period where simulation restores state from checkpoint data.

In the NOM simulation, the average restart time is about 11 seconds.

6.7 Checkpoint Interval for NOM Simulation

In our implementation, the checkpoint interval is calculated as follows:

- The average iterations to crash M is 5000 iterations.
- The average time to accomplish one time step is 1 second.
- The average checkpoint time c in time steps is thus 5 iterations.
- The average restart time in time steps r is thus 11 iterations.

The goal is to calculate the checkpoint interval in iterations, instead of time. From the above, the average checkpoint interval in iterations is the solution for the following equation:

$$\frac{x+5}{5000} = -\log(1 - \frac{x}{5000}) \tag{6.1}$$

Solving the above equation, we obtain the checkpoint interval is 224. This information about M, c and r are queried from the AWS data warehouse. Once the checkpoint interval is calculated, it is inserted into the Model table. When a new simulation is executed, the checkpoint interval is further retrieved from the Model table.

6.8 Building the NOM Data Warehouse

Data warehousing has been the platform for use of technologies such as online analysis processing (OLAP), decision support system (DSS) and data mining (DM). It has proven to be an invaluable tool by integrating information across the enterprise for decision making. The success of data warehousing in business motivates us to investigate and explore its use in the area of scientific simulation. Scientific simulations generate a huge amount of data, which motivates us to believe that data warehousing is a useful tool to better analyze and use the simulation data. In this paper, we describe our experience in building data warehouses for the scientific simulation project on modeling behavior of natural organic matter (NOM). We also describe the front end applications to visualize the simulation datasets. We believe this simulation data warehousing framework can be modified and generalized for other scientific simulation areas.

6.8.1 Introduction

Scientific simulations has been increasingly applied to solve a variety of scientific problems. Domains such as environmental science, in particular, benefit from this capability. As a demonstratible example, we have developed a series of models to simulate the behavior of natural organic matter (NOM). These models can play a critical role in understanding and predicting the properties of NOM over time as it evolves from precursor molecules to eventual mineralization [13].

However, running such simulations can require a large number of computing resources and produce terabytes of data. Effective storage and retrieval methods must be devised to cope with these volumes of data. Querying the simulation results is an enormously expensive task, especially for cross-simulation analysis. The success of data warehousing in the business world motivates us to think of data warehousing as a platform to store and analyze scientific simulation data. In this paper, we describe in detail the implementation and maintenance of a scientific data warehouse for environmental simulations. The simulation data warehousing framework can be easily modified and generalized for other areas of scientific simulation.

The simulation generated data was originally stored on local disks of simulation servers. However, there are many drawbacks of storing data on local disks. First of all, the data are inefficiently managed via text files and it is hard to query text files for data analysis. Secondly, there is no guarantee of data integrity. Thirdly, since data analysis must be done on local machines, collaboration among researchers was inhibited. Fourthly, simulation model verification and validation process requires parameter sweeping and cross simulation data analysis, and it is very difficult to accomplish without data warehouse support. Such obstacles hindered the possibility of scientists outside of computer science department to collect and analyze simulation data. With these drawbacks in mind, we design a data warehouse for

simulation data management and data analysis.

6.8.2 Background and Related Work

During the past years, the database researchers have been active in modeling and managing high dimensional scientific data [20, 83, 1, 25]. [83] describes data models to support data generated from physical experiments. [20] discusses the feasibility of building data stream systems for online analytical processing (OLAP). [1] compares scientific simulation data with data streams and argues that simulation data is a special case of data stream. Based on this argument, [1] builds the AQSim (Ad-hoc Queries for Simulation data) system. [25] develops an interface for data warehouse through a web-based GUI and let users query data. It uses a Java back end with JDBC connection to translate user commands into SQL queries and return the results to the users. [124] presents a comprehensive data warehouse framework which encompasses imaging and nom-imaging information in supporting disease management and research. The implementation is based on a Java CORBA and web-based architecture.

There are also efforts such as [54], to describe integration of metadata tools and data services systems for web-based management of large-scale scientific simulation data. However, none of the above related work provides sufficient details to implement and manage data warehouse specific to facilitate scientific simulations. A data warehouse can grow quickly to terabytes in size. Without satisfactory query performance, it is of little use. We believe it's worthwhile to point out the issues related to implementation and management of scientific simulation data warehouses to achieve excellent query performance. A well-tested commercial database such as Oracle and IBM DB2 is suitable to be the underlying database for building scientific data warehouses. In this paper, we use Oracle9i Release 2 to build data warehouses

because of its robustness and popularity.

6.8.3 Building Data Warehouses for Scientific Simulations

A data warehouse is a database that is designed for query and analysis rather than transaction processing. A data warehouse environment includes an extraction, transportation, transformation and loading (ETL) solution, online analytical processing (OLAP) and data mining capabilities [91]. A common way of introducing data warehousing is to refer to the characteristics of a data warehousing by W. Inmon [63]: subject oriented, integrated, nonvolatile and time variant.

In a data warehouse, the primary activity is querying data. The only update activity occurs when new data is loaded. Data warehouses are designed for quick retrieval. Information is often derived from other data, by rolling up data into summaries, drilling down to details, or looking for patterns and trends [53]. In an online transaction processing (OLTP) system, database schemas are designed using entity relation (ER) diagram. Normalization is used to ensure consistency and remove redundancy. In order to optimize performance for a data warehouse, a new data model other than ER diagram is needed. Kimball [74] introduced the star schema and dimensional modeling techniques. The dimensional approach organizes data into fact and dimensional tables. This process of designing dimensional and fact tables is called a logical design. Once the logical design is done, it has to be converted to a physical representation that optimizes performance and automates management. The rest of this section is organized as follows: we first discuss the requirement gathering process to build a successful data warehouse for scientific simulations; then we present the details to logically and physically build the data warehouse; after that we investigate and explore methods to automate data warehouse management; and finally we present web applications to visualize query and

data analysis results.

6.8.4 Requirements Gathering

All system designers know that you must have sound requirements in order to build any system, including data warehouses. Every project methodology includes a section on requirements gathering. The primary purpose of gathering requirements is to understand what the scientists needs from their data analysis environment. The data warehouse designer must understand the basic purpose of the scientists and the challenges facing them.

In an OLTP environment, system design life cycle is the common methodology to build an operational system. The correct requirements can be gathered with proper analysis in the beginning. And the end user requirements are often very clear. In a data warehousing environment, however, it is usually not possible to gather requirements at the beginning, because the end users often do not know how they will use the data. This is especially true for scientists. Without a prototype of data warehouse, the scientists do not know what can be done by the data warehouse and what they expect the data warehouse to accomplish for them. Therefore, it is not possible to start building the data warehouse with all requirements in hand. The goal is therefore to be flexible enough to deal with dynamically changing needs from scientists, and the data warehouse must be modified in an iterative way.

Although we can not gather all requirements from scientists to start with. We can still keep some initial requirements in mind. Such requirements include what queries need available. Such queries include "what is the evolutionary relationship between molecule density and time".

Although data warehousing has matured as a technology over the last few years, many challenges exist as the dynamic research environment changes. The data

warehouse must be built for performance and reliability and must be able to scale with increasing data volume and more users. As the data warehouse is growing to terabytes in size, with increasingly higher availability requirements, it is critical to maintain good performance for large number of geographically distributed scientists. The ability to visualize data warehouse reports on the Web makes information easily available to scientists. With the average user only staying on a Web page five seconds, it is vital to make reports up-to-date and available instantaneously. Therefore, reports must be summarized, precalculated and cached [53].

6.8.5 Building the Warehouse

Building a data warehouse involves extracting data from operational database systems and sometimes combining the data from flat files, transforming it into a uniform format and loading it into the database. Operational data must be extracted from the source operational systems and copied to the staging area - a temporary location where data is integrated, cleansed, transformed, and standardized for the warehouse. Once the data is transformed, it is ready to be transported and loaded into the warehouse.

Simulation Data Warehousing Architecture

The simulation data warehousing framework is based on multi-tiered architecture, as shown in Figure 6.3, which consists of the web applications for visualization, simulation servers on which simulations generate simulation data in flat files or in operational databases, the data staging area and the data warehouse.

- Simulation Server Pool - Simulation data is the major data source for the simulation data warehouse. Simulations run on one of the servers inside the simulation server pool. A load balancing algorithm is used to distribute simulation jobs to the simulation server pool. Be aware that long-running simulations

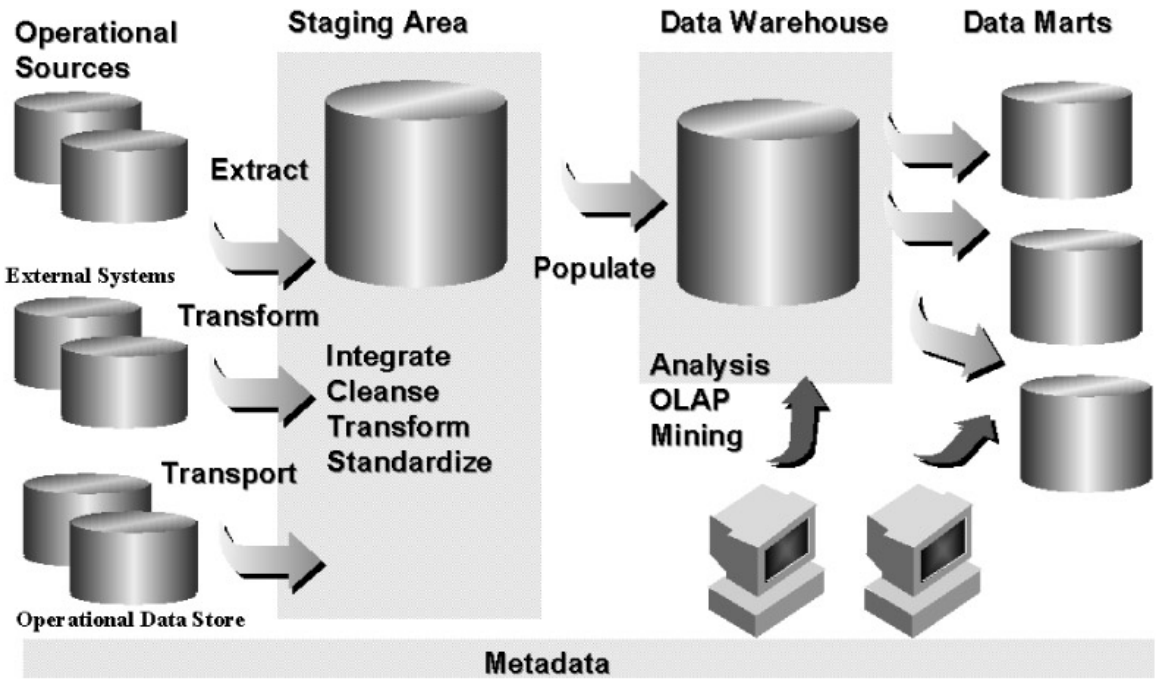

Figure 6.3. The multi-tiered simulation data warehousing architecture

are very likely to crash despite careful testing and debugging. Checkpointing seems to be the solution for long-running simulations. The details for load balancing and checkpointing/restarting are beyond the scope of this paper. Interested users may refer to our previous paper [60]. Data generated from the simulation are initially stored as text file on the local simulation server. The data are also loaded into an operational database system for user queries.

- Operational Data Sources - However, this operational database system is not capable of large-scale data analysis for simulation model verification, validation and finally for prediction. And data may need to be transformed before loading into the data warehouse. In our project, the operational databases reside on Red Hat Enterprise Linux Advanced Servers running Oracle9i in a private subnet.

- Staging Area - Data staging area is a temporary storage location for data that need to be integrated, cleansed, transformed and standardized. In our project, we develop PL/SQL packages and programs to do the work. One

restriction on the staging area is that it does not provide query and data analysis capabilities. These capabilities are available in data warehouses.

- Data Warehouse - Simulation data are loaded into the data warehouse using SQL*Loader and transportable tablespaces. The data warehouse is an Oracle9i database running on a Redhat Enterprise Linux Advanced Server. It provides services such as data analysis and visualization. The Oracle9i data warehouse provides SQL for aggregation, analysis, reporting, OLAP and data mining services. The data warehouse can also be linked to statistical software such as SPSS for hypothesis testing.

Design Considerations

The primary goal of a data warehouse is for query and data analysis. Therefore, we should focus on ensuring that queries are processed quickly. However, the data warehouse should also be designed to be easy to manage. The management issues include backup and recovery, loading new data, aggregating new data, indexing and archiving data. We briefly discuss methods that are useful to model the data warehousing for easy management and good performance.

In a data warehouse environment, the modeling methodology is dimensional modeling, instead of entity relationship modeling. The dimensional model views data from a different perspective and describe data using dimensions and facts. For example, we use dimensional modeling to design a data warehouse for the Reaction-Batch model, where Submission, Time, Environment and Molecule are dimensional tables and ReactionBatch is the fact table which records information for each submission, each time step, each environmental variable and molecule distribution.

During the new data loading phase, the expected loading time can be precalculated based on the amount of data and the degree of parallelism. Summary management can be used to create summary tables of aggregated data. There are many techniques that can be used to improve query performance. A classical of

which is to create various number of indexes based on the construction of query statements.

Another important consideration is the physical placement of data. Query performance can be improved if the number of I/O can be reduced. A big fact table is partitioned into a number of smaller tables and each of which is placed on separate I/O devices that are configured for bandwidth instead of capacity. Since the paritioning reduce the number of I/O contentions, query performance benefits from partitioning the fact tables.

Implementing the Data Warehouse

The target physical machine is a dual 3.0 Xeon processor Red Hat Enterprise Linux Advanced Server with several RAID 1 and RAID 5 systems on which the data warehouse has been implemented using Oracle9i Release 2.

It takes a few steps to create the data warehouse, shown as follows:

1. Create the starting Oracle database - This starting database can be created either manually or using the database configuration assistant. Our simulation data warehouse was named "db1".

2. Create the tablespaces and data files - A tablespace is created for all the dimensional tables, and another tablespace is created for indexes for all the dimensional tables. One tablespace suffices for the dimensional tables since there are small in size. A separate tablespace is created for each partition of the fact table. The fact table is partitioned based on the range of simulation_id, which is the identifier for each distinct simulation. Ideally these tablespace should reside on distinct hard disks. If not feasible, however, the hard disks can be used in a round-robin fashion. Also, a separate tablespace is create for local indexes of each partition. The following shows a snapshot of the SQL code to create these tablespaces. Note that the tablespaces for tables and the tablespaces for indexes reside on separate hard disks so that I/O contentions can be reduced and thus performance query performance is improved.

```
-- tablespace for dimensions
create tablespace nom_dim
 data file
'/data00/db1/dimensions.dbf'
 size 128M
```

```
 autoextend on default storage
 (initial 16M next 16M pctincrease 0
  maxextents unlimited);
create tablespace nom_dim_idx
 data file
 '/data01/db1/dimensions_idx.dbf'
 size 32M
 autoextend on default storage
 (initial 16M next 16M pctincrease 0
  maxextents unlimited);

-- tablespace for fact tables
-- for the first 300 simulations
create tablespace nom_fact100
 data file
'/data02/db1/fact100.dbf'
 size 128M
 autoextend on default storage
 (initial 16M next 16M pctincrease 0
  maxextents unlimited);
create tablespace nom_fact100_idx
 data file
 '/data01/db1/fact100_idx.dbf'
 size 32M
 autoextend on default storage
 (initial 16M next 16M pctincrease 0
  maxextents unlimited);
create tablespace nom_fact200
 data file
'/data02/db1/fact200.dbf'
 size 128M
 autoextend on default storage
 (initial 16M next 16M pctincrease 0
  maxextents unlimited);
create tablespace nom_fact200_idx
 data file
 '/data01/db1/fact200_idx.dbf'
 size 32M
 autoextend on default storage
 (initial 16M next 16M pctincrease 0
```

```
  maxextents unlimited);
create tablespace nom_fact300
 data file
'/data02/db1/fact300.dbf'
 size 128M
 autoextend on default storage
 (initial 16M next 16M pctincrease 0
  maxextents unlimited);
create tablespace nom_fact300_idx
 data file
 '/data01/db1/fact300_idx.dbf'
 size 32M
 autoextend on default storage
 (initial 16M next 16M pctincrease 0
  maxextents unlimited);
```

When more and more simulations are completed, new tablespaces for new partitions and indexes are created automatically.

3. Create the tables, constraints and indexes - The dimensional tables are created in the nom_dimensions tablespace and the indexes for the dimensional tablespace are created in the nom_dimensions_idx tablespace. The fact tables contain millions of rows and thus are partitioned based on the range of simulation_id. The following code snapshot shows the creation of the fact table ReactionBatch:

```
create table ReactionBatch
(simulation_id number not null,
 ...
) partition by range (simulation_id)
 ( partition fact100
    values less than 100
    pctfree 0 pctused 99
    storage (initial 16M next 16M
     pctincrease 0 )
    tablespace fact100,
   partition fact200
    values less than 200
    pctfree 0 pctused 99
    storage (initial 16M next 16M
     pctincrease 0 )
    tablespace fact200,
```

```
partition fact300
 values less than 300
 pctfree 0 pctused 99
 storage (initial 16M next 16M
  pctincrease 0 )
 tablespace fact300);
```

Indexes for these partitions are created correspondingly and reside on the appropriate tablespaces created before.

4. Define the security constraints - System and object privileges are granted to the users of the data warehouse so that the users can create summaries for aggregated data.
5. Analyze the tables and indexes after finishing data loading - The final step to create a data warehouse is to analyze the tables and indexes so that the cost-based optimizer can take into effect improve the query performance.

Once the data warehouse is created, data in the operational systems and flat files must be loaded. Populating a data warehouse involves all of the tasks related to getting the data, cleansing and transforming the data to the right format, loading the data into the data warehouse, and preparing data for analysis.

One of the most popular tools to loading data is SQL*Loader. In our implementation, we use SQL*Loader to load simulation data into the data warehouse. In a simulation data warehousing environment, the data transforming and loading processes can be automated because simulation data fits in predefined format. Automatic procedures are created to inspect the log files of SQL*Loader. If errors exist in the log file, the data is reloaded.

Optimizing the Warehouse

Simulation data warehouse is primary used to organize data for analysis and prediction queries. It is very easy for a simulation data warehouse grow to several terabytes in size. Without satisfactory query performance, a large data warehouse is of no use, therefore, it is extremely important for the data warehouse to retrieve

and process large amount of data efficiently. There are scientific data warehousing projects as indicated in the literature review section. But none of them discussed about performance of the data warehouses. We believe methods to improve query performance is critical in simulation data warehouses. We look at several techniques including data partitioning, indexing, query optimization automatic memory management and summary management.

Data warehouses tend to read large amounts of data to answer queries and hence it is important to use indexes. Indexes should be built on columns that are often part of the selection criteria of a query statement. Unlike OLTP systems, which have mostly update queries, data warehouse can have more indexes than an OLTP system.

Very large tables and indexes can be divided into smaller, more manageable partitions. Partitioning the data and indexes makes it possible for data management operations including queries to be performed at the partition level. It also allows queries to take the advantage of parallel processing.

There are different ways to retrieve the same set of rows in the data warehouse. An index could be used to locate the rows or a full table scan could be used to locate the same rows. One of most common queries in a data warehouse is the star query in which each of the dimension tables is joined to the fact table using the primary-key/foreign-key relationship. Creating appropriate indexes on the dimension tables and fact table makes it possible for the query to use star transformation, an algorithm for optimizing star queries.

Queries in the data warehouse need a significant amount of memory for sort and join operations. Various initialization parameters related to memory management must be tuned to get good performance. By taking the advantage of the automatic memory management feature of Oracle9i Release 2, we can relieve the burden of

memory tuning for each query to the system.

A common technique used in data warehouse is to precompute and store results of frequent queries, especially for queries involving aggregation. An example of such a query is the average molecular density over time for 100 consecutive simulations. As multiple scientists interested in the same query over time, the result would be aggregated over and over again for each user. Rather than wasting computing resources re-executing such query repeatedly, the result could be precalculated and saved in a table. Such tables are called summaries or summary tables. As new data is loaded into the simulation data warehouse, the data in the summary tables are not synchronized. In order to bring the summary tables up-to-date, it must be refreshed. The Summary Management feature provided in Oracle9i can keep summaries up-to-date automatically.

A well-designed data warehouse can deliver excellent performance for large queries that process large amounts of data.

6.8.6 Managing the Warehouse

Once the data warehouse has been created and populated with data, it is important to ensure it is correctly managed. Hard disks, memory and CPU resources must be managed. The management issues include backup and recovery, performance tuning, security maintenance and space usage monitoring.

Our data warehouse has a hot standby data warehouse residing on a different machine in the same private subnet. Once data is loaded in the data warehouse, the data is automatically transfered to the standby data warehouse. In this manner, the standby data warehouse can take the role of the primary data warehouse in case of failure.

The data warehouse should be tuned just like any other database systems, plus

the tasks such as improving data load time, improving query response, improving refreshing performance and improving maintenance time. Since the primary goal of a data warehouse is for query and data analysis, the first thought on performance tuning is to improve query performance. In a data warehousing environment, one of the best ways to improve query performance is to create materialized views. Provided the query is used frequently, with the existence of a materialized view, the query can be executed by retrieving the results from the materialized view rather than the detail table.

Security is less important in simulation data warehouses than business data warehouses. But since the data warehouse is in use for scientists geographically distributed, it is still important to make sure the data is correct when transferring over the Internet. We can use the GRANT and REVOKE commands to specify explicitly who can access the tables. Another techniques we use is to create roles, assign privileges to the roles and then grant the roles to the users.

Another very important management issue is to know how much space is available in the data warehouse. One technique we use to avoid running out of space is to create the data files with autoextend and unlimited extents. However, this won't help if the disks actually fill up. We created space monitoring scripts so that when the disk space usage is approaching some threshold (e.g., 90%), some operations such as purging or compression are performanced automatically.

6.8.7 Visualization

The advancement of internet technologies make it possible to publish the data from the data warehouse to the Internet. We use the Oracle9i Application server and JFreeChart [67] to create web applications to deliver reports to the scientists through the Web. The web applications use JDBC to retrieve data from the data

warehouse and use JFreeChart to convert data into graphical reports. JFreeChart is a free Java class library for generating charts including pie charts, bar charts, line and area charts, time series charts, and so on.

One important usage of our simulation data warehouse is to help scientists study the time-dependent evolution of NOM in the environment. There are many time-dependent features for the NOM molecules, such as the number of adsorptions, the number of precursor molecules, the molecular density and so on. See Appendix B for some visualization screenshots.

6.8.8 Summary

We have presented a framework to build scientific simulation data warehouses and visualization software for supporting environmental research. The significance of this work is more practical than theoretical. The contribution of this work is to establish the viability of data warehouses in managing and analyzing scientific simulation data.

Our experience in building scientific simulation data warehouses include the following:

- Simulation data warehouses can grow quickly in size. Performance and management become key issues. In this paper, we not only described the design and implementation of the data warehouse, but also presented methods to improve performance and automate management.
- Data analysis software such as SPSS, and data visualization applications can be incorporated to the data warehouse. The development of scientific simulation data warehousing solutions makes a close collaborative effort among environmental scientists by providing efficient data storage, information retrieval, data analysis and visualization tools.
- A well designed data warehouse can speed up data mining processes because data can be efficiently queried in generally time-consuming data mining

processes.

- The data warehouse can be easily extended to incorporate experimental data for the purpose of simulation model verification and validation. The data analysis SQL functions provided by the data warehouse are important vehicles for scientists to advance research and understanding in the area of environmental science.

We are currently using the data warehouse for simulation model verification and validation, data mining and hypothesis testing. We are also gathering feedback from our end users (environmental scientists) to make iterative improvements and incorporate more data analysis procedures for testing and evaluation purposes. Keep in mind that building a data warehouse is not a short term project, but a long term iterative process, and therefore the data warehouse can always be improved. Future work will explore the data warehouse to better serve scientists and test the generality of the data warehousing framework by using it for other scientific researches. Such a scientific research is to understand the open source software development phenomenon [38].

6.9 Summary

In this chapter, we have described the agent-based stochastic simulation approach to model the behavior of natural organic matter (NOM). The simulation has been implemented using Java/Swarm/RePast. To make the simulation accessible to geographically distributed scientists, we have designed and implemented a portal using the best practice Model-View-Controller based architecture. The portal is deployed to a self-manageable computing infrastructure. We also described our experience on building the NOM data warehouse for better data analysis and provided a case study using data mining to study the NOM simulation data.

CHAPTER 7

CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

It is believed that autonomic computing will be the next era of computing. Autonomic computing provides a natural platform for web-based simulations. In this dissertation, we have explored current information technologies that are in favor of autonomic computing. We also find autonomic computing is useful in the field of web-based simulations.

7.1 Summary of Achievements

Many scientific simulations are large programs which despite careful debugging and testing will probably contain errors when deployed to the Web for use. Based on the assumption that such scientific simulations do contain errors and the underlying computing systems do fail due to hardware or software errors, we investigate and explore robust methods for building reliable systems to support web-based scientific simulations with the presence of such errors. We have presented a framework to build autonomic web-based simulation (AWS). AWS cannot come from nowhere. Certain requirements must be satisfied during the developing process of AWS. More precisely, the simulation should be able to checkpoint and restart; J2SE 5.0 monitoring and management APIs are very helpful to make simulations internally self-manageable. A self-manageable computing infrastructure is necessary to host web-based simulations. AWS strives to achieve the four features presented in the Vision of Autonomic

Computing [72]: self-configuring, self-optimizing, self-healing and self-protecting.

In this dissertation, we have proposed and compared three models to simulate the lifecycle of scientific simulations. We can draw the following conclusions about checkpointing scientific simulations:

- The choice of checkpoint interval is independent of the restart time r; however, the predicted total execution time is exponentially dependent on r. Therefore, the total execution time can be reduced dramatically if a failure can be detected and restart can be accomplished quickly.
- An under-predicted checkpoint interval results in longer total execution time than an equivalent over-predicted one; and therefore, we would rather to choose a larger checkpoint interval if it's not possible or too difficult to calculate the optimal.

In this dissertation, we have also evaluated a set of simple and yet efficient data cleansing methods through approximate string join. The goal of the approximate string join is to find (almost) all pairs of strings such that their distance is below a certain threshold. The approximate string join methods are based on a three step approach. We first map the database of strings into points in an euclidean space, then use approximate matrix multiplication to find approximately all pairs of close points. These pairs of close points are further evaluated using the string distance to filter false positives. We implement this method in a commercial database. Experiments show that this approach is both effective and efficient.

7.2 Future Directions

According to the the CIO Today Magazine (March 2004), "Many CIOs are spending good portions of their IT budgets on improving their existing infrastructure". In other words, there is a focus on increasing utilization. Autonomic computing promises self-manageable systems. Our future research will focus on ways to build autonomic enterprise information systems.

Some premilary thoughts on autonomic information systems:

- Self-configurable enterprise systems: global optimization seems to be a promising method to auto-tune enterprise systems performance related parameters. More investigation on current global optimization methods will be conducted and find most efficient existing or new methods for global optimization. Note that efficiency is the most prominant consideration in configuring enterprise systems.

- Self-optimizing enterprise systems: Control theory seems to be a good method for self-optimization. Future work requires a good understanding of control theory. In general a control system is composed of a controller and a controlled system. The controller controls the controlled system via feed-forward control strategy or feed-back control strategy. For example, the controlled system could be a J2EE application server and the controller regulates performance.

APPENDIX A

CODE USED IN CHAPTER 3

```
public class Checkpoint {

    public Checkpoint() {}

    private int crash_now(int M) {
        return (int) (-M * Math.log(Math.random()));
    }

    public int total_time(int N, int M, int r, int x, int c)

    //M is the average time before crash, and
    //x is the checkpoint interval, and
    //r is the restart time, and
    //c is the checkpoint time, and
    //N is the total time required if no crash and no checkpoint
    {

        int steps_completed = 0;
        int total_steps     = 0; //to be returned
        int next_total_steps = 0;
        int point_of_failure = crash_now(M);

        //System.out.println("Crash at time step: "+crash_time);
        int     next_checkpoint   = 0;
        int     execution_segment = 0;
        boolean segment_success   = true;
        int     redo              = 0;

        while(steps_completed < N) {
            if((steps_completed + x) < N) {
                execution_segment = x + c; //xc-segment: x+c
            } else {
```

```
                execution_segment = N - steps_completed + c;
            }

            if( !segment_success) {
                execution_segment += r;   //rxc-segment: r + x + c
            }

            next_total_steps = total_steps + execution_segment;

            if(next_total_steps < point_of_failure) {
                segment_success = true;
                steps_completed += x;
                total_steps     += execution_segment;
            } else     // crashed
            {
                segment_success  = false;
                total_steps      = point_of_failure;
                point_of_failure += crash_now(M);
            }
        }

        return total_steps;
    }

    public double
      avg_total(int N, int M, int r, int x, int c, int runs) {

        int    sum_total = 0;
        double avg_total = 0.0;

        for(int i = 0; i < runs; i++) {
            sum_total += total_time(N, M, r, x, c);
        }

        if(runs > 0) {
            avg_total = (double) sum_total / runs;
        }

        return avg_total;
    }
}
```

APPENDIX B

SELECTED SCREEN SHOTS

B.1 Select Screen Shots for NOM Simulation

Figure B.1. NOM homepage

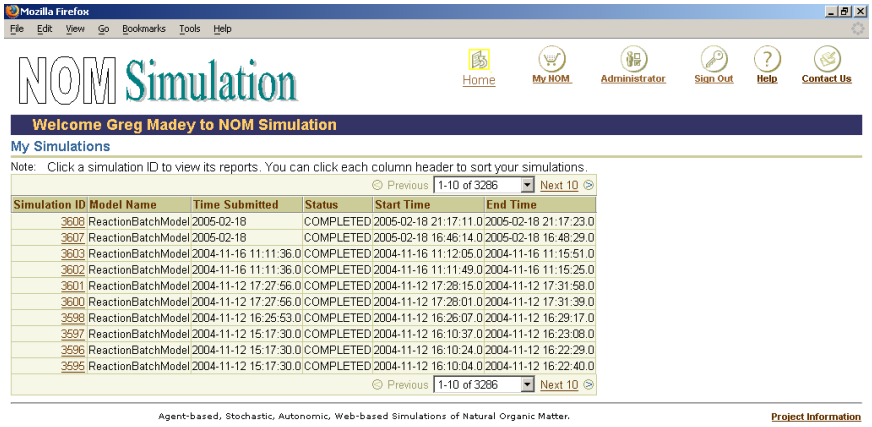

Figure B.2. NOM mynom page

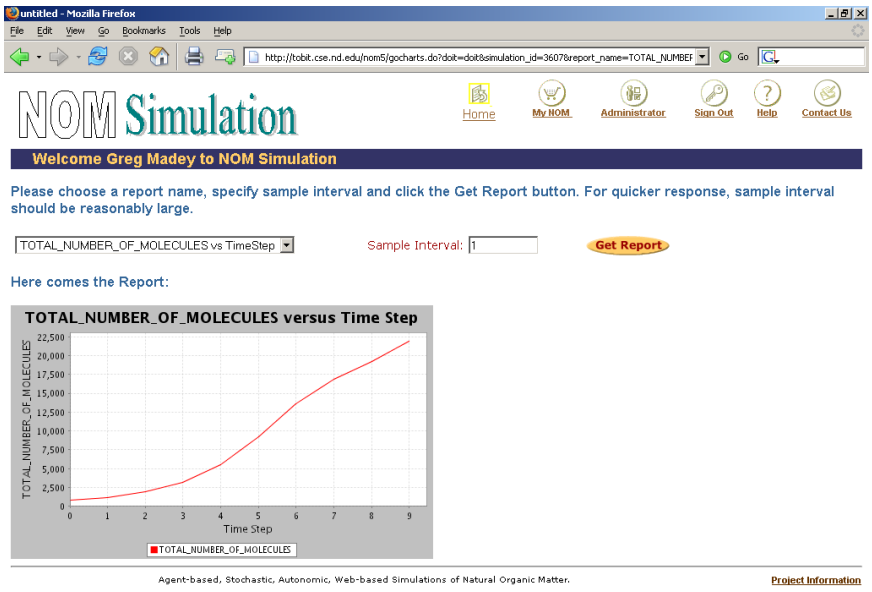

Figure B.3. NOM report for simulation 3607

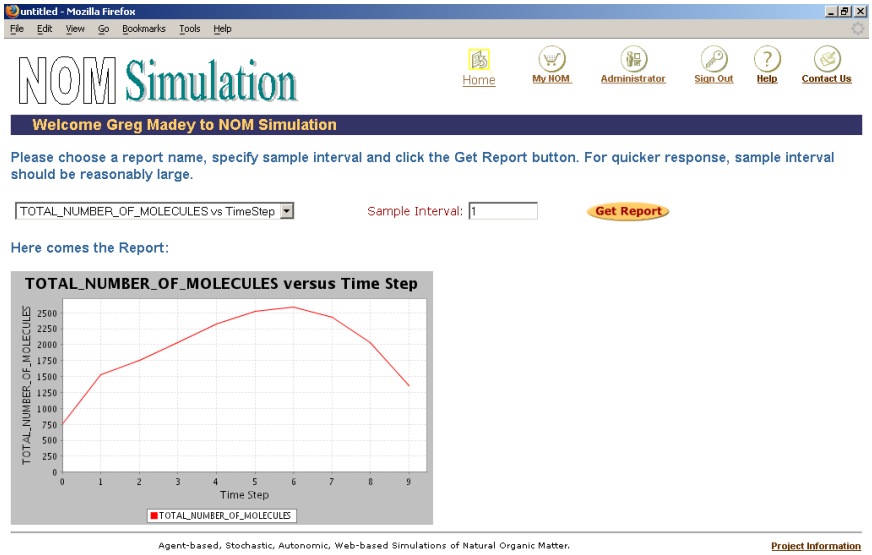

Figure B.4. NOM report for simulation 3608

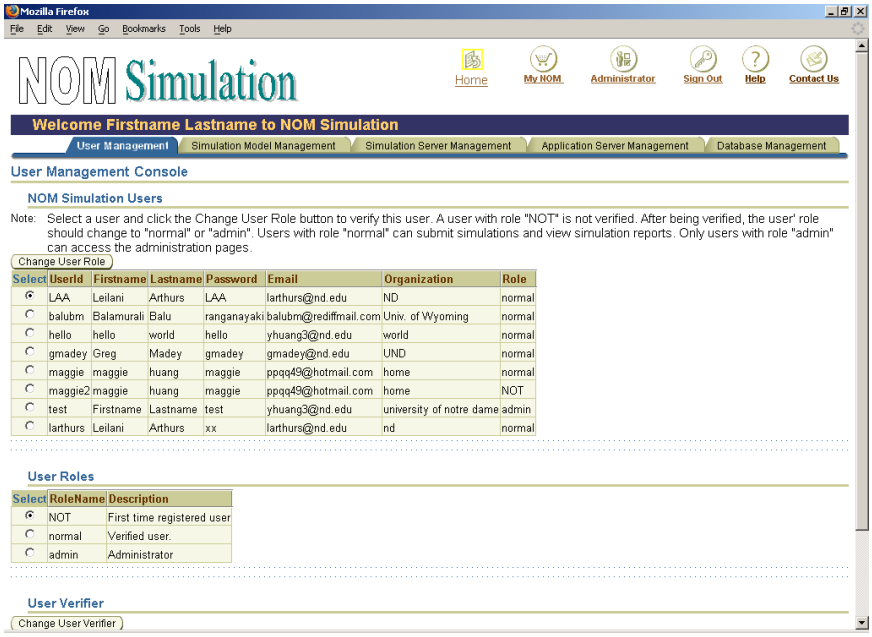

Figure B.5. NOM management page

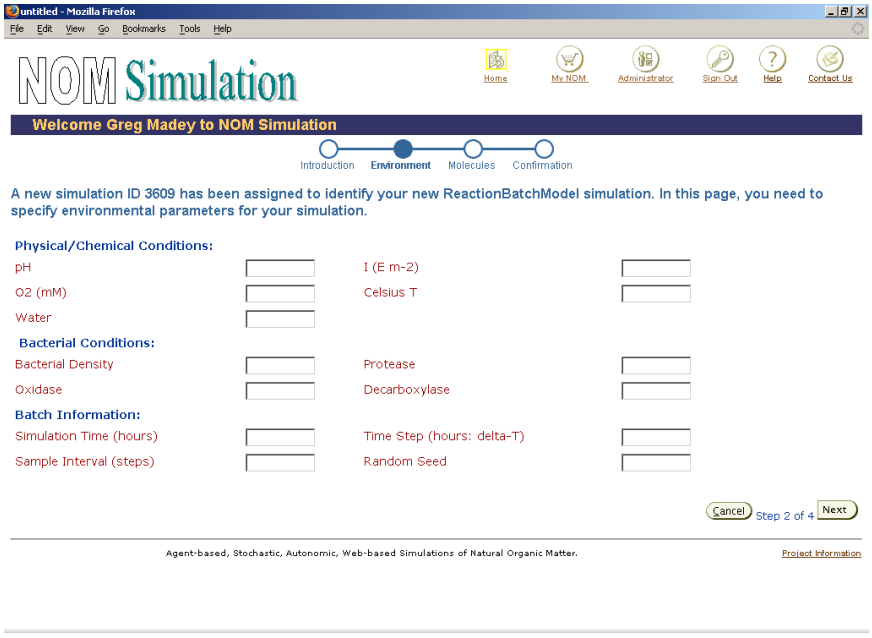

Figure B.6. NOM simulation input page

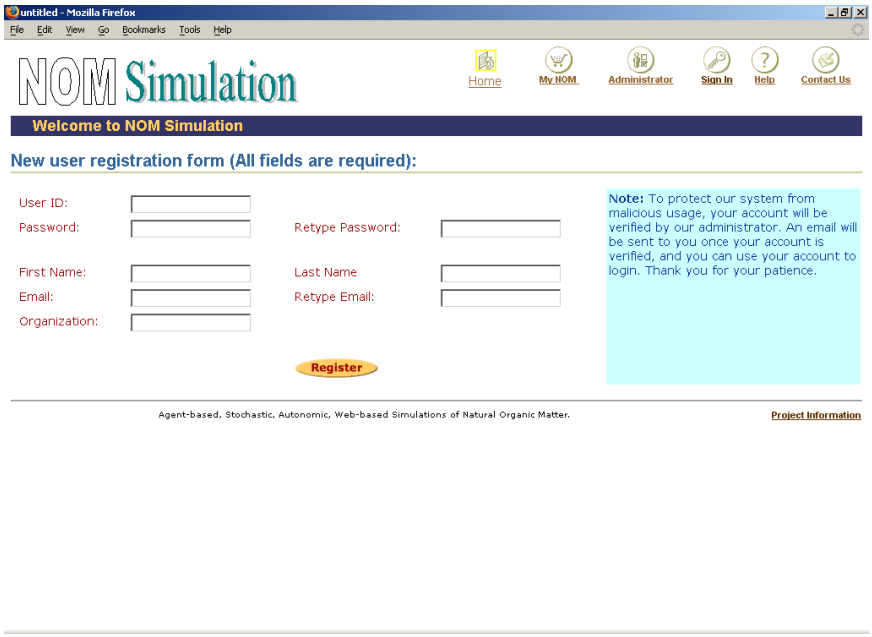

Figure B.7. NOM sign up page

APPENDIX C

MONITORING APPLICATION SERVER AND DATABASE SERVER AVAILABILITY

```
#!/bin/sh

LOGDIR=/home/yhuang3/research/monitor/log
PID=$LOGDIR/monitor.pid
AS_PID=$LOGDIR/monitor_as.pid
AS_RESPONSE=$LOGDIR/as_response
DB_RESPONSE=$LOGDIR/db_response
DB_PID=$LOGDIR/monitor_db.pid

ASUPDOWN=$LOGDIR/asupdown
DBUPDOWN=$LOGDIR/dbupdown

AS_SERVER=http://tobit.cse.nd.edu
DB_SERVER=nom/nom@db1
# DB_SERVER=yhuang3/yhuang3@joy

email_me(){
  echo "$2" |mail -s "$1" -c gmadey@nd.edu yhuang3@nd.edu
}


monitor_as(){
  previous_as_status=‘cat $ASUPDOWN‘
  echo "previous as status "$previous_as_status >> $LOGDIR/status.log

  (
    wget -q -s -O - $AS_SERVER &
    as_pid=$!
    echo $as_pid > $AS_PID
    { sleep 5; kill $as_pid; } &
```

```
    sleep 1
    wait $as_pid
  ) | head -1 |cut -d' ' -f2 >$AS_RESPONSE
  resp=`cat $AS_RESPONSE`
  echo "response code "$resp >> $LOGDIR/status.log

  if [ $resp == "200" ]; then
    echo 1 > $ASUPDOWN
  else
    echo 0 > $ASUPDOWN
  fi
  as_status=`cat $ASUPDOWN`
  echo "as status "$as_status >> $LOGDIR/status.log

  if [ $previous_as_status == "1" ]; then
    if [ $as_status == "0" ]; then
      echo "mail me" >> $LOGDIR/status.log
      subject="AS down"
      text="AS is down as of "`date`
      email_me "$subject" "$text"
    fi
  fi
}

monitor_db(){
  previous_db_status=`cat $DBUPDOWN`
  echo "previous db status "$previous_db_status >> $LOGDIR/status.log

  (
    sqlplus $DB_SERVER @db1.sql &
    db_pid=$!
    echo $db_pid > $DB_PID
    { sleep 5; kill $db_pid; } &
    sleep 1
    wait $db_pid
  ) | grep "Connected to:" >$DB_RESPONSE
  resp=`cat $DB_RESPONSE`
  echo "response code "$resp >> $LOGDIR/status.log

  if [ ${#resp} -ne 0 ]; then
    echo 1 > $DBUPDOWN
  else
    echo 0 > $DBUPDOWN
  fi
```

```
  db_status=‘cat $DBUPDOWN‘
  echo "db status "$db_status >> $LOGDIR/status.log

  if [ $previous_db_status == "1" ]; then
    if [ $db_status == "0" ]; then
      echo "mail me" >> $LOGDIR/status.log
      subject="DB1 down"
      text="DB1 is down as of "‘date‘
      email_me "$subject" "$text"
    fi
  fi
}

start(){
  touch $PID
  echo 1 >$ASUPDOWN
  echo 1 >$DBUPDOWN

  { { while true;
      do
        # monitor application server using wget
        monitor_as
        # monitor database server using sqlplus
        monitor_db
        sleep 300
      done
    } > /dev/null 2>&1 </dev/null &
  pid=$!
  echo $pid > $PID
  }&
  echo "Monitor is started."
}

stop(){
  if [ -f $PID ]; then
    for i in ‘cat $PID‘
    do
      kill -9 $i
    done
    rm -rf $PID
  fi
  echo "Monitor is shutdown."
}
```

```
status(){
  if [ -f $PID ]; then
    pid=‘cat $PID‘
    echo "Monitor is running with pid="$pid
  else
    echo "Monitor is shutdown."
  fi
}

restart(){
  stop
  start
}

case "$1" in
start)
        start
        ;;
stop)
        stop
        ;;
status)
        status
        ;;
restart)
        restart
        ;;
*)
        echo "Invalid argument."
        ;;
esac
```

APPENDIX D

EDIT DISTANCE IMPLEMENATION IN PL/SQL

```
create or replace function ed(s in varchar2, t in varchar2)

    return number
    deterministic
  as
    s_len number :=nvl(length(s), 0);
    t_len number :=nvl(length(t), 0);
    type thenumbers is table of number index by binary_integer;
    col_to_left thenumbers;
    cur_col thenumbers;
    v_cost number:=0;
  begin
    if s_len=0 then
      return t_len;
    elsif t_len=0 then
      return s_len;
    else
      for j in 0..t_len loop
        col_to_left (j) := j;
      end loop;
      for i in 1..s_len loop
        cur_col(0) := i;
        for j in 1..t_len loop
          if substr(s, i, 1) = substr(t, j, 1) then
             v_cost :=0;
          else
             v_cost :=1;
          end if;
          cur_col(j) := least (cur_col(j-1)+1,
                               col_to_left(j) +1,
                               col_to_left(j-1) + v_cost);
        end loop;
```

```
        for j in 0..t_len loop
          col_to_left (j) :=cur_col(j);
        end loop;
      end loop;
    end if;
    return cur_col(t_len);
  end;
/
```

APPENDIX E

NOM REACTION BATCH MODEL DATA

```
SQL> desc reactionbatchmodel
 Name                                      Null?    Type
 ----------------------------------------- -------- -------
 SIMULATION_ID                             NOT NULL NUMBER
 TOTAL_NUMBER_OF_MOLECULES                          NUMBER
 TOTAL_MOLECULAR_WEIGHT                             NUMBER
 MWN                                                NUMBER
 MWW                                                NUMBER
 Z_AVERAGE                                          NUMBER
 EQUIVALENT_WEIGHT                                  NUMBER
 PERCENT_AROMATIC                                   NUMBER
 TOTAL_MASS_C                                       NUMBER
 TOTAL_MASS_H                                       NUMBER
 TOTAL_MASS_N                                       NUMBER
 TOTAL_MASS_O                                       NUMBER
 TOTAL_MASS_S                                       NUMBER
 TOTAL_MASS_P                                       NUMBER
 PERCENT_C                                          NUMBER
 PERCENT_H                                          NUMBER
 PERCENT_N                                          NUMBER
 PERCENT_O                                          NUMBER
 PERCENT_S                                          NUMBER
 PERCENT_P                                          NUMBER
 ESTER_CONDENSATION                                 NUMBER
 ESTER_HYDROLYSIS                                   NUMBER
 AMIDE_HYDROLYSIS                                   NUMBER
 MICROBIAL_UPTAKE                                   NUMBER
 DEHYDRATION                                        NUMBER
 CC_STRONG_OXIDATION                                NUMBER
 CC_WEAK_OXIDATION                                  NUMBER
 ALCOHOL_OXIDATION                                  NUMBER
```

ALDEHYDE_OXIDATION	NUMBER
DECARBOXYLATION	NUMBER
HYDRATION	NUMBER
ALDOL_CONDENSATION	NUMBER

APPENDIX F

DERIVATION OF MODEL III

Let $T = T_{total}(x)$ for simplicity. Then

$$\begin{aligned} T &= N + \frac{Nc}{x} + \left(M + \frac{x+c}{1 - e^{\frac{x+c}{M}}} + r \right) \frac{T}{M} e^{-\frac{x+c+r}{M}} \\ &+ \left(M + \frac{x+c+r}{1 - e^{\frac{x+c+r}{M}}} \right) \frac{T}{M} \left(1 - e^{-\frac{x+c+r}{M}} \right) \end{aligned}$$

which reduces to

$$0 = N + \frac{Nc}{x} + \frac{T}{M} \left(\left(\frac{x+c}{1 - e^{\frac{x+c}{M}}} \right) - (x+c) \right) e^{-\frac{x+c+r}{M}}$$

Divide both sides by $(x + c)$, we obtain

$$0 = \frac{N}{x} + \frac{T}{M} \left(\left(\frac{1}{1 - e^{\frac{x+c}{M}}} \right) - 1 \right) e^{-\frac{x+c+r}{M}}$$

Thus

$$T = NMe^{\frac{r}{M}} \frac{e^{\frac{x+c}{M}} - 1}{x}$$

APPENDIX G

PROOF OF LEMMA 2.4

Proof. Without lose of generality, it suffices to prove that

$$T(x) = \frac{e^{x+c} - 1}{x}, 0 < x < 1 \tag{G.1}$$

has the following property: for any t such that $0 < x^* - t < x^* + t < 1$, we have $T(x^* - t) > T(x^* + t)$ where $x^* = \arg\min_{0<x<1} T(x)$. In fact, since $x^* = \arg\min_{0<x<1} T(x)$, we have $e^{x^*+c} = \frac{1}{1-x^*}$. Let

$$h(t) = (T(x^* - t) - T(x^* + t))(x^{*2} - t^2) = t(e^{-t} + e^t - 2) - x^*(e^t - e^{-t} - 2t)$$

Let $g(t) = e^t - e^{-t} - 2t$, then $g(0) = 0$ and $g'(t) = e^t + e^{-t} - 2 > 0$. Therefore, $g(t) > 0$. Note that $x^* < 1 - t$ and $0 < t < \frac{1}{2}$. Hence,

$$h(t) > t(e^{-t} + e^t - 2) - (1 - t)(e^t - e^{-t} - 2t) = e^{-t} - e^t + 2te^t - 2t^2$$

Let

$$p(t) = e^{-t} - e^t + 2te^t - 2t^2, 0 < t < \frac{1}{2}. \tag{G.2}$$

Then $p(0) = 0$ and

$$\begin{aligned} p'(t) &= -e^{-t} - e^t + 2e^t + 2te^t - 4t \\ &= (e^t - e^{-t} - 2t) + 2t(e^t - 1) \\ &= g(t) + 2t(e^t - 1) > 0 \end{aligned}$$

Therefore, $p(t) > 0$. Hence $h(t) > 0$ and finally $T(x^* - x) > T(x^* + t)$. □

APPENDIX H

IMPLEMENTATION OF AUTONOMIC AGENTS ON SIMULATION SERVERS

```
#!/bin/sh

DIR=/export/disk1/users/nom/agents
HOST=‘hostname | tr -d "sim"‘
PIDFILE=$DIR/agent${HOST}.pid
URL=nom/nom@db1

start(){
 if [ -f $PIDFILE ]; then
  echo "Agent seems to be running. Try to restart."
  restart
  return
 fi
{
{
while true
do
newserver
loadavg
checkjob
check_crashed_job
failure_detect
sleep 3
done
} >/dev/null 2>&1 </dev/null &

pid=$!
echo $pid > $PIDFILE
} &

}
```

```
stop(){
 if [ ! -f $PIDFILE ]; then
  echo "Agent is not running."
  return
 fi
 local line
 read line < $PIDFILE
 kill $line || echo stop agent
 rm -f $PIDFILE || echo remove agent pid
}

restart(){
 stop
 start
}

newserver(){
 sqlplus $URL >/dev/null <<!
  insert into servers (server_name) values ($HOST);
  commit;
  exit
!
}

loadavg()
{
 LOADAVG=‘cat /proc/loadavg | (read u v w x y; echo $u)‘
 sqlplus $URL >/dev/null <<!
  update servers set loadavg=${LOADAVG}, now=sysdate, status='UP'
  where server_name=${HOST};
  commit;
  exit
!
}

checkjob()
{
 SPOOL=$DIR/check${HOST}.txt

sqlplus $URL >/dev/null <<EOF
  set head off
  set feedback off
  set termout off
  spool $SPOOL
```

```
  select * from dispatcher where appserver=$HOST;
  spool off
  exit;
EOF

SED="sed -e '/SQL>/d' -e '/^[    ]*$/d' $SPOOL"
eval $SED | while read SIMULATION_ID SERVER SIMULATOR TIMEWAIT
do
  #echo $SIMULATION_ID
  #echo $SERVER
  #echo $SIMULATOR

  echo "java -jar ${SIMULATOR}.jar $SIMULATION_ID"
  java -jar ${DIR}/${SIMULATOR}.jar $SIMULATION_ID batch 0 \
  > ${DIR}/output/${SIMULATION_ID}.out &
  PID=$!

  # delete dispatcher
  # update submissions
  # update simulations
sqlplus $URL >/dev/null <<EOF
  delete dispatcher;
  update submissions set confirmed='D'
         where simulation_id=$SIMULATION_ID;
  insert into simulations (simulation_id, model_name, server_name,
         status, starttime) values ($SIMULATION_ID, '$SIMULATOR',
         '$SERVER', 'EXECUTING',  sysdate);
  insert into simulation_pid (simulation_id, pid, server) values (
         $SIMULATION_ID, $PID, $HOST);
  commit;
  exit;
EOF
done
}

check_crashed_job()
{
SPOOL2=${DIR}/check_crashed${HOST}.txt

sqlplus $URL >/dev/null <<EOF
  set head off
  set feedback off
  set termout off
  spool $SPOOL2
```

```
  select * from dispatcher_restart where appserver=$HOST;
  spool off
  exit;
EOF

SED="sed -e '/SQL>/d' -e '/^[    ]*$/d' $SPOOL2"
eval $SED | while read SIMULATION_ID SERVER SIMULATOR TIMEWAIT
do
  echo "java -jar ${SIMULATOR}.jar $SIMULATION_ID"
  java -jar ${DIR}/${SIMULATOR}.jar $SIMULATION_ID batch 0 \
  > output/${SIMULATION_ID}.out &
  PID=$!

  # delete dispatcher
  # update submissions
  # update simulations
sqlplus $URL >/dev/null <<EOF
  delete dispatcher_restart;
  update crashed_simulations set confirmed='D'
         where simulation_id=$SIMULATION_ID;
  update simulations set status='RESTARTED'
         where simulation_id=$SIMULATION_ID;
  update simulation_pid set pid=$PID, server=$HOST
         where simulation_id=$SIMULATION_ID;
  commit;
  exit;
EOF
done
}

failure_detect()
{
SPOOL3=${DIR}/failure_detector${HOST}.txt

sqlplus $URL >/dev/null <<EOF
  set head off
  set feedback off
  set termout off
  spool $SPOOL3
  select s.simulation_id, s.model_name, sp.pid from
         SIMULATIONS s, SIMULATION_PID sp
         where s.simulation_id = sp.simulation_id and
         s.status != 'COMPLETED' and server=$HOST;
  spool off
```

```
  exit;
EOF

SED="sed -e '/SQL>/d' -e '/^[    ]*$/d' $SPOOL3"
eval $SED | while read SIMULATION_ID MODEL_NAME PID
do
echo $SIMULATION_ID $MODEL_NAME $PID
STATUS=`ps -fu $USER |grep -v grep | grep " $PID "`
if [ $? != 0 ]; then
sqlplus $URL <<EOF
  update simulations set status='CRASHED'
         where simulation_id=$SIMULATION_ID;
  insert into  crashed_simulations (simulation_id, model_name,
         time_submitted, confirmed) values  (
         $SIMULATION_ID, '$MODEL_NAME', sysdate, 'Y');
  update crashed_simulations set time_submitted=sysdate,
         confirmed='Y' where simulation_id=$SIMULATION_ID;
  commit;
  exit;
EOF
fi
done
}

case $1 in
start)
  start
  ;;
stop)
  stop
  ;;
restart)
  restart
  ;;
*)
  echo "usage: daemon.sh [start|stop]"
  ;;
esac
```

BIBLIOGRAPHY

[1] G. Abdulla, T. Critchlow, and W. Arrighi. Simulation data as data streams. *SIGMOD Record*, 33(1):89–94, 2004.

[2] M. Agarwal and M. Parashar. Enabling autonomic compositions in grid environments. In *Proceedings of the Fourth International Workshop on Grid Computing*, pages 34–41, 2003.

[3] G.R. Aiken, D.M. McKnight, R.L. Wershaw, and P. MacCarthy. *Humic substrances in soil, sediment, and water - geochemistry, isolation and characterization.* John Wiley and Sons, New York, 1985.

[4] J. Ametller, S. Robles, and J.A. Ortega-Ruiz. Self-protected mobile agents. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent systems*, pages 362–367, 2004.

[5] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, 2002.

[6] J. Appavoo, K. Hui, C.A.N. Soules, R.W. Wisniewski, D.M. Da Silva, O. Krieger, M.A. Auslander, D.J. Edelsohn, B. Gamsa, G.R. Gander, P. McKenney, M. Ostrowski, B. Rosenburg, M. Stumm, and J. Xenidis. Enabling autonomic behavior in systems software with hot swapping. *IBM Systems Journal*, 42(1):60–76, 2003.

[7] L. Arthurs, P.A. Maurice, X. Xiang, R. Kennedy, and G. Madey. Agent-based stochastic simulation of natural organic matter adsorption and mobility in soils. In *11th International Symposium on Water-Rock Interaction*, 2004.

[8] M. Bilendo and R.J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proc. ACM-SIGKDD*, 2003.

[9] G.A. Bolcer and G. Kaiser. Swap: Leveraging the web to manage workflow. *IEEE Internet Computing*, 3(1):85–88, 1999.

[10] J. Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. In *Israel Journal of Mathematics, v52*, pages 46–52, 1985.

[11] J.M. Bull, L.A. Smith, L. Pottage, and R. Freeman. Benchmarking java against c and fortran for scientific applications. In *Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande*, pages 97–105, 2001.

[12] S. Cabaniss. *Modeling and stochastic simulation of NOM reactions.* http://www.nd.edu/ nom, 2002.

[13] S. Cabaniss, G. Madey, P. Maurice, L. Leff, Y. Huang, and X. Xiang. Stochastic synthesis model for the evolution of natural organic matter. In *225th American Chemical Society National Meeting*, 2003.

[14] S.E. Cabaniss, Q. Zhou, P.A. Maurice, Y.P. Chi, and G.R. Aiken. A log-normal distribution model for the molecular weight of aquatic fulvic acids. In *Environ. Sci. Technol. 34*, pages 1103–1109, 2000.

[15] B. Carpenter, V. Getov, G. Judd, A. Skjellum, and G. Fox. *MPJ: MPI-like message passing for Java*. Concurrency: Practice and Experience, 2000.

[16] A. Cerpa and D. Estrin. Ascent: adaptive self-configuring sensor networks topologies. In *UCLA Technical Report UCLA/SCDTR-01-0009*, 2001.

[17] K.M. Chandy. A survey of analytic models for rollback and recovery strategies. *Computer*, 8(5):40–47, 1975.

[18] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technologies. *SIDMOD Record*, 1997.

[19] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleansing. In *ACM SIDMOD*, 2003.

[20] Y. Chen, G. Dong, J. Han, J. Pei, B.W. Wah, and J. Wang. Online analytical processing stream data: is it feasible? In *Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2002.

[21] E. Cohen and D.D. Lewis. Approximate matrix multiplication for pattern recognition tasks. In *ACM-SIAM Symposium on Discrete Algorithms (SODA97)*, 1997.

[22] W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *ACM SIGKDD*, 2002.

[23] W.W. Cohen and J.Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proc. ACM-SIGKDD*, 2002.

[24] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *ACM-SIAM Symposium on Discrete Algorithms (SODA02)*, 2002.

[25] T. Critchlow, K. Fidelis, M. Ganesh, R. Musick, and T. Slezak. Datafoundry: Information management for scientific data. *IEEE Transactions on Information Technology in Biomedicine*, 4(1):52–57, 2000.

[26] C. Dabrowski and K. Mills. Understanding self-healing in service-discovery systems. In *Proceedings of the first workshop on self-healing systems*, pages 15–20, 2002.

[27] C. Dabrowski, K. Mills, and A. Rukhin. Performance of service-discovery architectures in response to node failures. In *Proceedings of the International Conference on Software Engineering Research and Practice*, pages 95–101, 2003.

[28] E.M. Dashofy, A. van der Hoek, and R.N. Taylor. Understanding self-healing in service-discovery systems. In *Proceedings of the first workshop on self-healing systems*, pages 21–26, 2002.

[29] Y. Diao, J.L. Hellerstein, S. Parekh, and J.P. Bigus. Managing web server performance with autotune agents. *IBM Systems Journal*, 42(1):136–149, 2003.

[30] P. Drineas and R. Kannan. Fast monte-carlo algorithms for approximate matrix multiplication. In *IEEE Foundations on Computer Science (FOCS)*, 2001.

[31] A. Duda. The effects of checkpointing on program execution times. *Information Processing Letters*, 16:221–229, 1983.

[32] S. Dutta. Building manageability using the management and monitoring apis to build application manageability. *JDJ*, 9(11):40–44, 2004.

[33] C. Faloutsos and K. Lin. Fastmap: a fast algorithm for indexing, data-mining and visulization of traditional and multimedia datasets. In *ACM SIDMOD*, 1995.

[34] W. Feng, D. Kandlur, D. Saha, and K. Shin. A self-configuring red gateway. In *Proceedings of IEEE INFOCOM*, 1999.

[35] L. Ferreira, V. Berstis, J. Armstrong, M. Kendzierski, A. Neukoetter, M. Takagi, R. Bing-Wo, A. Amir, R. Murakawa, O. Hernandez, J. Magowan, and N. Bieberstein. *Introduction to grid computing with Globus*. IBM Corporation, 2003.

[36] I. Foster and C. Kesselman. *The Grid 2: Blue prints for a new computing infrastructure*. Morgan Kaufmann, 2003.

[37] G. Fox. E-science meets computational science and information technology. *IEEE: Computing in Science and Engineering*, pages 84–85, 2002.

[38] Y. Gao, Y. Huang, and G. Madey. Data mining project history in open source communities. In *NAACSOS Conference*, 2004.

[39] E. Gelenbe and M. Hernandez. Optimum checkpoints with age dependent failures. *Acta Informatica*, 27:519–531, 1990.

[40] I. Georgiadis, J. Magee, and J. Kramer. Self-organizing software architectures for distributed systems. In *Proceedings of the 1st workshop on self-healing systems*, pages 33–38, 2004.

[41] V. Getov, G. von Laszewski, M. Philippsen, and I. Foster. Multiparadigm communications in java for grid computing. *Communications of ACM, Volume 14, Issue 10*, 2001.

[42] F. Glover and M. Laguna. *Tabu search: modern heuristic techniques for combinatorial problems*. Scientific Publications, Oxford, 1993.

[43] L. Golab and M.T. Ozsu. *Data Stream Management Issues - A Survey.* Technical Report, University of Waterloo, 2003.

[44] D.E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley, 1989.

[45] V. Grassi, L. Donatiello, and S. Tucci. On the optimal checkpointing of critical task and transaction-oriented systems. *IEEE Transactions on Software Engineering*, 18(1):72–77, 1992.

[46] L. Gravano and P.G. Ipeirotis. Using q-grams in a dbms for approximate string processing. In *IEEE Data Engineering Bulletin 24(4)*, pages 28–34, 2001.

[47] L. Gravano, P.G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an rdbms for web data integration. In *Proc. WWW2003*, 2003.

[48] J. Gray. What next?: A dozon information-technology research goals. *Journal of the ACM (JACM)*, 50(1):41–57, 2003.

[49] The Grinder. *The Grinder.* http://grinder.sourceforge.net/, 2004.

[50] J. Gross. *Agent-based modeling in ethnobiology: a brief introduction to outside.* http://www.tiem.utk.edu/ gross, 2002.

[51] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *Proc. ACM-SIGMOD*, 1995.

[52] G.R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. In *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 25, No. 5*, 2003.

[53] L. Hobbs. *Oracle9iR2 Data Warehousing.* Digital Press, Inc, 2003.

[54] V.P. Holmes, W.R. Johnson, and D.J. Miller. Integrating metadata tools with the data services archive to provide web-based management of large-scale scientific simulation data. In *Proceedings of the 37th Annual Simulation Symposium*, pages 72–79, 2004.

[55] G. Hristescu and M. Farach-Colton. Cluster-preserving embedding of proteins. In *technical report, Rutgers University*, 1999.

[56] Y. Huang and G. Madey. Data integration through approximate string joins. In *Proceedings of WWW (Alternative Track and Posters)*, 2004.

[57] Y. Huang and G. Madey. Towards autonomic computing for web-based simulation. In *Proceedings of Internation Conference on Cybernatics and Information Technologies, Systems and Applications*, 2004.

[58] Y. Huang and G. Madey. Autonomic web-based simulation. In *Proceedings of the 38th Annual Simulation Symposium*, 2005.

[59] Y. Huang and G. Madey. A data warehouse for autonomic web-based simulation. In *Proceedings of Internation Conference on Cybernatics and Information Technologies, Systems and Applications*, 2005.

[60] Y. Huang, X. Xiang, and G. Madey. A self manageable infrastructure for supporting web-based simulations. In *Proceedings of the 37th Annual Simulation Symposium*, pages 149–156, 2004.

[61] Y. Huang, X. Xiang, G. Madey, and S. Cabaniss. Agent-based scientific simulation. *IEEE Computing in Science and Engineering*, 7(1):22–29, 2005.

[62] IBM. *Autonomic computing manifesto.* http://www.research.ibm.com/-autonomic/manifesto/, 2003.

[63] W.H. Inmon. *Building the Data Warehouses.* Wiley Computer Publishing, 1996.

[64] B. Jacob, L. Ferreira, N. Bieberstein, C. Gilzean, J-Y. Birard, R. Strachowski, and S. Yu. *Enabling applications for grid computing with Globus.* IBM Corporation, 2003.

[65] J. Jann, L.M. Browning, and R.S. Burgula. Basic building blocks for autonomic computing on ibm pseries servers. *IBM Systems Journal*, 42(1):29–37, 2003.

[66] Java. *Java Management Extension.* http://java.sun.com/j2se/-1.5.0/docs/guide/management/, 2004.

[67] jfree.org. *JFreeChart.* http://jfree.org/jfreechart/index.html, 2003.

[68] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *Proc. 8th International Conference on Database Systems for Advanced Applications (DASFAA)*, 2003.

[69] R. Johnson. *Expert One-on-One: J2EE Design and Development.* Wrox Press, 2002.

[70] S. Joines, R. Willenborg, and K. Hygh. *Performance analysis for Java web sites.* Person Education, Inc, 2003.

[71] K. Kalbitz, S. Solinger, J.H. Park, B. Mchalzik, and E. Matzner. Controls on the dynamics of dissolved organic matter in soils:a review. In *Soil Sci.165*, pages 277–304, 2000.

[72] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[73] B. Khargharia, S. Hariri, M. Parashar, L. Ntaimo, and B. uk Kim. Vgrid: A framework for building autonomic applications. In *Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environment*, pages 19–27, 2003.

[74] R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite. *The Data Warehouse Lifecycle Toolkit.* John Wiley & Sons, Inc, 1998.

[75] J.A. Kohl and P.M. Papadopoulas. Efficient and flexible fault tolerance and migration of scientific simulations using cumulvs. In *Proceedings of the SIGMETRICS symposium on Parallel and distributed tools*, pages 60–71. ACM Press, 1998.

[76] V.G. Kulkarni, V.F. Nicola, and K.S. Trivedi. Effects of checkpointing and queuing on program performance. *Communications of Statistical Stochastic Models*, 6(4):615–648, 1990.

[77] S.W. Kwak, B.J. Chio, and B.K. Kim. An optimal checkpointing strategy for real time control systems under transient faults. *IEEE Transactions on Reliability*, 50(3):293–301, 2001.

[78] N. Koudas L. Gravano, P. Ipeirotis and D. Srivastava. Text join in an rdbms for web data integration. *Proc. 12th international WWW conf.*, 2003.

[79] G. Lanfranchi, P. Della Peruta, A. Perrone, and D. Calvanese. Toward a new landscape of systems management in an autonomic computing environment. *IBM Systems Journal*, 42(1):119–128, 2003.

[80] Y. Lin, B.R. Preiss, W.M. Loucks, and E.D. Lazowska. Selecting the checkpoint interval in time warp simulations. In *Proceedings of the 7th workshop on Parallel and Distributed Simulation*, pages 3–10, 1993.

[81] Y.B. Lin and E.D. Lazowska. Recuding the state saving overhead for time warp parallel simulation. *Technical Report 90-02-03, Department of Computer Science and Engineering, University of Washington*, 1990.

[82] Y. Ling, J. Mi, and X. Lin. A variational calculus approach to optimal checkpoint placement. *IEEE Transactions on Computers*, 50(7):699–707, 2001.

[83] D. Malon and E. May. Critical database technologies for high energe physics. In *Proceedings of the 23rd VLDB Conference*, 1997.

[84] V. Markl, G.M. Lohman, and V. Raman. Leo: An autonomic query optimizer for db2. *IBM Systems Journal*, 42(1):98–106, 2003.

[85] A. Maydanchik. Challenges of efficient data cleansing. In *DM Direct*, 1999.

[86] A.J. McAuley and K. Manousakis. Self-configuring networks. In *Proceedings of 4th Adv. Telecommun. and Info. Dist. Res. Conf.*, 2000.

[87] D.M. McKnight and G.R. Aiken. *Sources and age of humus*. Springer-Verlag, Berlin, 1985.

[88] Microsoft. *Dynamic Systems Initiative*. http://www.microsoft.com/-windowsserversystem/dsi/dsioverview.mspx, 2004.

[89] V.F. Nicola. *Checkpointing and the modeling of program execution time*. John Wiley & Sons, 1995.

[90] L.N. Norere and T.S. Shimizu. *Stochsim: modeling of stochastic biomolecular processes*. Bioinformatics, 2001.

[91] Oracle. *Oracle data warehousing guide*. http://www.oracle.com, 2003.

[92] YA. Palaniswamy and P.A. Wilsey. An analytical comparison of periodic checkpointing and incremental state saving. In *Proceedings of the 7th workshop on Parallel and Distributed Simulation*, pages 127–134, 1993.

[93] M.P. Papazoglou and D. Georgakopoulos. Service-oriented computing. *Communications of the ACM*, 46(10), 2003.

[94] H.V.D. Parunak, R. Savit, and R.L. Riolo. Agent-based modeling vs equation-based modeling: a case study and user's guide. In *Proceedings of multi-agent systems and agent-based simulation*, 1998.

[95] D. Patterson. *Recovery Oriented Computing (ROC): Motivation, Definition, Techniques and Case Studies*. Computer Science Technical Report UCB/CSD-02-1175, U.C. Berkeley, 2002.

[96] B.R. Preiss, I.D. MacIntyre, and W.M. Loucks. On the trade-off between time and space on optimistic parallel discrete-event simulation. In *Proceedings of the 6th workshop on Parallel and Distributed Simulation*, pages 3–10, 1992.

[97] D. Pyne. *Data preparation for data mining*. Morgan Kaufmann, 1999.

[98] M. Raghavachari, D. Reimer, and R.D. Johnson. The deployer's problem: configuring application server for performance and reliability. In *Proceedings of the international conference on software engineering*, pages 484–489, 2003.

[99] RePast. *http://repast.sourceforge.net*. 2003.

[100] H.E. Romeijn and R.L. Smith. A first order approximation to the optimum checkpoint interval. *Communications of ACM*, 17(9):530–531, 1974.

[101] H.E. Romeijn and R.L. Smith. Simulated annealing and adaptive search in global optimization. *Probability in the Engineering and Informational Sciences*, 8:571–590, 1994.

[102] R. Ronnagren and R. Ayani. Adaptive checkpointing in time warp. *ACM SIGSIM Simulation Digest*, 24(1):110–117, 1994.

[103] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proc. ACM-SIGKDD*, 2002.

[104] R.G. Sargent. Validation and verification of simulation models. In *Proceedings of the 31st Winter Simulation Conference*, 1999.

[105] G. Schreck, T. Schadler, C. Rustein, and A. Tseng. The fabric operating system. In *Technical report, Forrester*, 2003.

[106] M.D. Schroeder, A. D. Birrell, and M. Burrows. Autonet: a high-speed, self-configuring loca area network using point-to-point links. In *IEEE Journal on Selected Areas in Communications*, 1991.

[107] M.W. Shapiro. Self-healing in modern operating systems. *Queue*, 2(9):66–75, 2004.

[108] T.S. Shimizu, S.V. Aksenov, and D. Bray. *A spatially extended stochastic model of the bacterial chemotaxis signalling pathway.* Journal of Molecular Biology, 2003.

[109] T.S. Shimizu and D. Bray. *Foundations of systems biology.* Computational Cell Biology- - The stochastic approach, 2001.

[110] K.G. Shin, T. Lin, and Y. Lee. Optimal checkpointing of real-time tasks. *IEEE Transactions on Computers*, 36(11):519–531, 1987.

[111] S.K. Shrivastava and S.M. Wheater. Architectural support for dynamic reconfiguration of large scale distributed applications. In *Proceedings of the 4th International Conference on Configurable Distributed Systems*, pages 10–17, 1998.

[112] A. Singhal. Design of a data warehouse system for network/web services. In *Proceedings of Conference on Information and Knowledge Management*, pages 10–17, 2004.

[113] H.M. Soliman and A.S. Elmaghraby. An analytical model for hybrid checkpointing in time warp distributed simulation. *IEEE Transactions on Parallel and Distributed Systems*, 9(10):947–951, 1998.

[114] Struts. *http://struts.apache.org.* 2004.

[115] Swarm. *http://www.swarm.org.* 2002.

[116] A.N. Tantawi and M. Ruschitzka. Performance analysis of checkpointing strategies. In *Proceedings of the 1983 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, page 129, 1983.

[117] Y. Tao. *Large-scale network parameter configuration using on-line simulation framework.* Ph.D. Dissertation, Rensselaer Polytechnic Institute, 2003.

[118] S. Tejada, C.A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accurary object identification. In *Proc. ACM-SIGKDD*, 2002.

[119] G.K. Thiruvathukal. Java at middle age: Enabling java for computational science. *IEEE: Computing in Science and Engineering*, pages 74–84, 2002.

[120] M. Trofin. A self-optimizing application server design for enterprise java beans applications. In *Proceedings of the 18th annual ACM SIGPLAN*, pages 396–397, 2003.

[121] T.N. Truong. An integrated web-based grid-computing environment for research and education in computational science and engineering. In *Proceedings of the 37th Annual Simulation Symposium*, pages 143–148, 2004.

[122] O. Vormoor. Quick and easy interactive molecular dynamics using java3d. *IEEE: Computing in Science and Engineering*, pages 98–104, 2001.

[123] B.C. Williams and P.P. Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of AAAI*, 1996.

[124] S.T.C. Wong, K.S. Hoo, R.C. Knowlton, K.D. Laxer, X. Cao, R. A. Hawkins, W.P. Dillon, and R.L. Arenson. Design and applications of a multimodality image data warehouse framework. *Journal of the American Medical Informatics Association*, 9(3):239–254, 2002.

[125] T. Wu. A passive protected self-healing mesh network architecture and applications. *IEEE/ACM Transactions on Network (TON)*, 2(1):40–52, 1994.

[126] B. Xi, Z. Liu, M. Raghavachari, C.H. Xia, and L. Zhang. A smart hill-climbing algorithm for application server configuration. In *Proceedings of the international conference on the World Wide Web*, pages 484–489, 2004.