Architecture for a Web-based Environmental Research Support Portal

Yingping Huang
Department of Computer Science & Engineering
University of Notre Dame
Notre Dame, IN 46556
yhuang3@nd.edu

Xiaorong Xiang
Department of Computer Science & Engineering
University of Notre Dame
Notre Dame, IN 46556
xxiang1@nd.edu

Gregory Madey
Department of Computer Science & Engineering
University of Notre Dame
Notre Dame, IN 46556
gmadey@nd.edu

Abstract

In this paper, we describe the design and implementation of a multi-tier architecture to support environmental research portal. This architecture demonstrates the successful integration of web server, application server, database server, reports server, data warehousing, data mining and simulation models. It achieves high scalability through load balancing, and high reliability through simulation resuming. The load balancing feature is implemented based on simulation completion time prediction. The simulation resuming feature is implemented using Java transaction service (JTS/JTA). The architecture also integrates simulation sharing and other features to support scientific collaboration among scientists. A scientific simulation, NOM, is used to demonstrate the effectiveness of the architecture.

1. Introduction

Most simulation models currently available in the fields of scientific simulations run in stand-alone or traditional client-server architecture. Both of these models require installing software on user's computers. This presents a significant barriers due to incompatibility that complicates or prevents installations. Meanwhile, the stand alone or traditional client-server approach has some significant drawbacks: (1) lack of scalability since it's a single user program, (2)lack of collaboration features, such as simulation sharing among users, (3) lack of reliability, and (4) lack of centralized simulation management and data analysis.

To overcome these drawbacks, we present a multi-tier architecture such that users can run simulations and view simulation reports using their web browsers, such as Netscape and Internet Explorer. The users only need to specify inputs for their simulations, data analysis and reports are completed at the back end and delivered to the users through the reports server.

This architecture can serve multiple users and each user can run multiple simulations. Scalability and reliability are achieved through the load balancing and simulation resuming features, as we'll discuss them in detail later. It can serve as a template or guideline for future web-based simulation model design. That's our research goal of this paper.

To evaluate the effectiveness of this architecture, we employ the NOM simulation project, which is a multi-disciplinary project granted by NSF. The NOM (Natural Organic Matter) simulation program is written in Java, using agent-based stochastic simulation algorithms. Huge amount of data is generated by the program. Database technologies seems to be necessary to store and analyze the data. In the next sections, we'll demonstrate how data is stored and analyzed.

Since Internet based technologies have been mature, there are quite a few related works, [11] [15] [13]. The main contribution of our work is that we design and implement the architecture to support a web-based environmental research portal, with high scalability and reliability, automated data analysis and collaboration tools.

The rest of the paper is organized as follows: In section 2, we present a brief introduction to the technologies employed by the architecture. In section 3, the overall architecture is demonstrated. In section 4 and 5, we show the details to implement load balancing and simulation resuming. In section 6, we discuss reports and collaboration tools. Conclusion and future work are given in the last section.

2. Technologies

The web architecture employs advanced J2EE (Java 2 Enterprise Edition), XML and Oracle technologies. Let's give a brief introduction to these technologies.

2.1. J2EE

According to Sun Microsystems, the Java 2 Enterprise Edition (J2EE) utilizes the unified, scalable and platform independent Java language and uses a component-based approach to design, develop, assemble and deploy enterprise applications [7]. J2EE consists of separate technology areas for building successful systems utilizing these technologies. A quick review of some of the available technologies that are applied to this architecture is given below.

- **JDBC** The Java DataBase Connection (JDBC) technology lets Java programs access almost any tabular or relational data source. JDBC will be used everywhere in our system, including the core simulation programs which need database access to store simulation data, and web interfaces which provide simulation configurations and reports.
- **JTA and JTS** The Java Transaction Service and Java Transaction API ensure data integrity by enforcing strict rules for accessing and manipulating data. JTA enables a distributed transactional system or application to access a transaction manager, as defined by JTS. JTA and JTS will be used to implement simulation-resuming, which simply means that when a simulation is crashed or terminated for any reason, it can be resumed. The reason for JTA and JTS is that we need to span updates to multiple databases.
- **Servlet, JSP and EJB** The Java Servlet API is a container managed, stateful alternative to Common Gateway Interface (CGI) applications. A servlet is used to receive HTTP requests, generate dynamic data, and deliver HTTP response. The Java Server Page lets developers embed java code into HTML pages. JSPs are compiled into Java Servlets at runtime. An enterprise java bean is a server-side component that encapsulates the business logic of an application. We use servlets, JSP and EJB to design and implement the interfaces of the web applications.

Among several choices of application servers, such as BEA WebLogic, Sun One, IBM WebSphere, JBOSS, and OC4J (Oracle9iAS Container for J2EE), we use OC4J as the J2EE platform in our architecture because one type of reports generated by XSQL needs to run on OC4J. OC4J can be downloaded from www.Oracle.com.

2.2. XML

XML, which stands for the "Extensible Markup Language", defines a universal standard for data exchange. It provides a set of rules that enable the structure inherent in data to be easily encoded and interpreted by human-readable text format [16]. XML will be used for simulation configuration and simulation reports generation. Reports in XML format can be further processed by users. The XML reports are generated using XSQL (XML SQL from Oracle).

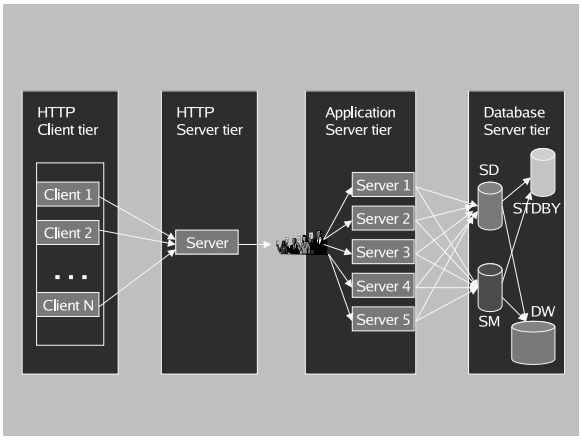

Figure 1. The Multi-tier Architecture

2.3. Oracle RDBMS, Data Warehousing and Data Mining

We use Oracle as the database tier of the multi-tier architecture. According to Inmon [6], a data warehouse is a database with the following features: subject oriented (defined by subject matter), integrated (data from disparate sources), nonvolatile (rarely updated), and time variant (grow over time). The star schemas are designed in the data warehouse for simulation reports. Data mining refers to extracting or mining knowledge from large amounts of data. Data generated by many simulations are transformed and feed into the data mining software to discover interesting patterns.

3. Architecture Description

The web-based environmental research portal is built on a multi-tier architecture, as shown in Figure 1. Users on the HTTP client tier communicates with the HTTP server tier, which routes the incoming requests to the application servers through the load balancer. The application servers host scientific simulations and connect to the database servers to store simulation data for analysis. Data analysis reports are delivered using JSP and XML to the users' browsers through the Reports Server.

We use Oracle in the database tier and OC4J in the application server tier in our architecture.

Let's first talk about how the database tier is organized. To improve overall performance and reliability, we use four

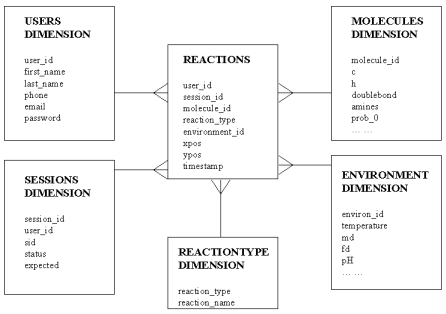

Figure 2. The star schema in DW

database servers , namely, SD (simulation data), SM (simulation management), DW (data warehouse) and STDBY (standby databases).

SD is used to store simple reports data when the simulation is running. Data generated by the simulation program is summarized by the program and inserted into SD using JDBC.SM stores information about users, jobs, checkpoints. A checkpoint stores all information about a simulation at the time of the checkpointing. A simulation can be restarted using the checkpoint data. DW is used to archive simulation data. Data generated by simulations are first written in local files, when the simulation completes, the local files are loaded into DW using SQL*Loader. Data analysis takes place in DW and more reports are generated automatically. It's designed using standard data warehousing technologies, such as star schema and summary tables. Figure 2 shows an example of star schema in DW. A standby database (STDBY) is created for each of SD and SM to improve reliability. Once the primary database are down for any reason, the standby databases can take the role and resume the applications.

When more and more simulations are submitted simultaneously, the application server seems to be the bottleneck of the architecture. Load balancing techniques can be employed to eliminate this bottleneck. In the next section, we'll show how load-balancing can be implemented. We provide two algorithms to implement load-balancing. One is to choose an application server with smallest load average. The other is to predict simulation completion time using a data mining approach. Based on the prediction, the resulting application server will be chosen such that the simulation can be completed in shortest time potentially. The first algorithm has been implemented in the architecture, the second one is still under investigation.

To guarantee that a submitted simulation can be guaranteed to be completed, we employ the Java Transaction Service/API, to checkpoint a simulation periodically. The goal of checkpointing is that when a simulation crashes during execution, it can be restarted using the checkpointing data. A checkpoint is a distributed transaction which spans the two databases SD and SM. The two transaction branches can be executed in parallel. This improves performance and reliability.

4. Load Balancing

The goal of load-balancing is to distribute simulations evenly among application servers. There are many load-balancing schemes available in the literature, but none of them can guarantee an even distribution if applied directly, since our simulation jobs are time-consuming and I/O bound. Available load-balancing algorithms include the following, to name a few:

- Round Robin: new jobs are assigned to the application servers sequentially. For example, suppose we have 6 jobs to be assigned to 5 application servers, then job 1 is assigned to application server 1, job 2 is assigned to application server 2, and so on. Job 6 is assigned to application server 1 again. Round Robin is the simplest load-balancing algorithm. It does not take the execution time of jobs into account.

- Shortest Queue (SQ) [14]: a new job is assigned to an application server with the smallest number of jobs running. It does not take job complexity and server status into account.

- Shortest Expected Delay (SED) [2]: A job is assigned to the application server which takes the least expected time to complete. In [2], the expected completion time simply counts the number of jobs currently waiting in the queue. In our situation, we need a different mechanism to predict the expected completion time of a new simulation.

We propose two load-balancing algorithms. One is to simply assign the simulation to an application server which has the smallest load average. Another approach is to predict the completion time of a simulation based on simulation configuration, machine loads, and history of simulation jobs. We use an instance learning approach. This algorithm tries to load-balancing the application servers by assigning a simulation job to an application server which is predicted to complete the job in shortest time. Note that this algorithm is different from SED. In SED, the completion time prediction is simply derived from number of waiting jobs.

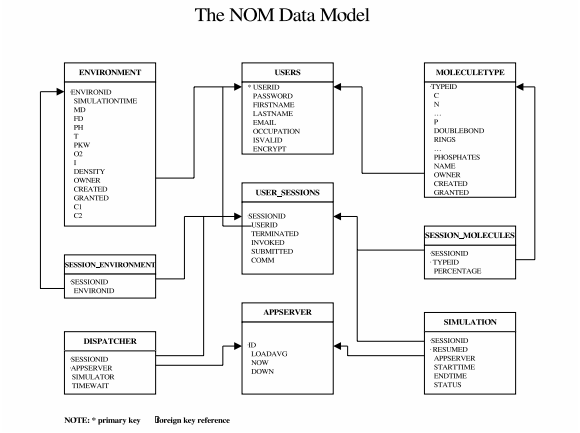

Figure 3. Data model for the NOM portal

4.1. Implementing the Simple Load-balancing Algorithm

We use Bourne shell scripts and Oracle PL/SQL to implement load-balancing based on the load averages of the application servers. The machine load average, measuring how may processes are waiting to be executed, can be obtained by running the "uptime" command on the local host. The load average is updated every 5 seconds. A simulation is assigned to one of the application servers which has the lowest load average.

If two or more simulations are assigned in a short time interval, they might be assigned to the same application server since the load average is not updated yet in the short time interval. To prevent this situation, we choose to assign simulations at most once in every 10 seconds to ensure that the load average is updated after one simulation is assigned. Next, we'll show how to implement this simple load balancing algorithm.

A data model is designed and implemented in the SM database. Figure 3 shows the model to support load balancing.

Two shell scripts are implemented. One is called dispatch.sh which is running as a daemon on the machine hosting the HTTP server, the other is called iagent.sh which is running on each application server. Figure 4 and Figure 5 shows the two shell scripts.

The daemon dispatch.sh does two things: dispatch simulation jobs to application servers, and send KEEPALIVE messages to all running simulations. The daemon iagent.sh does three things: upload local host load average (the 1 minute load average), check simulation jobs, and send ACK messages to the dispatcher. The dispatching algorithm is implemented using PL/SQL. First of all, it checks the table DISPATCHER to see if any records exists. If any, it just increase the value of the field TIMEWAIT by 1. The

```
#!/bin/sh

while [ true ]
do
sqlplus qiuqiu/qiuqiu@sm >/dev/null <<EOF
exec dispatch;
exit;
EOF

/export/home/yhuang3/daemon/keepalive.sh
sleep 10
done
```

Figure 4. Daemons running on HTTP server and application servers

```
#!/bin/sh

if [ $# -ne 1 ]; then
  echo "usage: iagent <appserver>"
  exit
fi

while [ true ]
do
  /export/home/yhuang3/daemon/loadavg.sh $1
  /export/home/yhuang3/daemon/checkjob.sh $1
  sleep 5
done
```

Figure 5. Daemons running on HTTP server and application servers

field TIMEWAIT counts the number of times the simulation job is checked, for the purpose of detecting whether the assigned application server is still on line. When the value of TIMEWAIT is larger than 2, the corresponding record is deleted and the assigned application server is marked as DOWN. Then all currently running simulations on this application server will be marked as crashed. The dispatcher then checks the table SIMULATION for crashed simulations. If any, the crashed simulation with smallest SESSIONID will be executed. This ensures that crashed simulations have higher priority than newer simulations. If there is no crashed simulations, the dispatcher checks the table USER_SESSIONS for newly submitted simulations. The new simulation with smallest SESSIONID will be executed. The application server with smallest load average will be chosen as the application server to run the selected simulation. If the smallest loadavg is larger than a threshold, say 10, then all application servers are busy, and the simulation is not executed. The dispatcher puts the selected simulation job into the table DISPATCHER, which stores at most one row at any time.

The iagent daemons on application servers check the table DISPATCHER to see whether there is a job for them. The iagent daemon removes the record in the table DISPATCHER if it sees a simulation job for it. Mean while, a new record is inserted into the table SIMULATION. The iagent daemons check the table DISPATCHER every five seconds. The iagent also checks the whether the simulation on the local host is completed. If yes, another another process which loads the simulation data into the data warehouse DW and analyzes the data is invoked.

The simple algorithm always pick the application server with the smallest load average to execute a new simulation. We tested the algorithm by submitting many simulations. All application servers running these simulations had roughly the same load averages. Therefore, this load balancing algorithm achieved reasonably good load balance.

The major drawback of this simple algorithms is that it only takes the load average into account. But the completion time of a simulation is related to not only CPU (load average), but also memory, disk I/O, network I/O and database I/O. Thus more sophisticated algorithms are required to accurately predict the execution time of simulations. The next section describes our effort on this.

4.2. A new load-balancing algorithm

The goal of the new load-balancing algorithm is to find an application server such that the job can be completed earliest.

To make efficient use of the computer resources, we propose an algorithm which tries to find the application server such that new simulation jobs can be completed in shortest time. The basic idea is to predict the job completion time for each application server, based on the history of jobs and the status of the application servers.

Execution time prediction has been studied in fields of operating systems and parallel computing, [9] [12] [1] [10] [4] [5] [8] [3]. But these approaches does not count for the complexity of scientific simulations. Next we'll present our new load-balancing algorithm. We first estimate the completion time of a new simulation for each application server based on the load averages of the application servers and the amount of data which needs to be stored to the database.

There are tables to keep track of load average history, number of simulations, availability of application servers and predicted completion time of each running simulations.

The execution time of a simulation depends on the requests of iterations (comes from the simulation configuration), denoted by I, machine load average, denoted by $L(t)$, a function of time, and the average amount of data generated each iteration, denoted by P, the number of units of data storage. 8192 bytes is the Oracle data block size for each database. There is much overhead for each data block, for example, the data block header, the row identifiers, column identifiers. After careful computation of the overhead, we choose 7000 bytes as the unit of data storage. We propose the following formula to compute the expected execution time:

$$ExpTime(t) = I * (\frac{C_1}{t} \int_0^t L(s)ds + C_2 P) \qquad (1)$$

where C_1 and C_2 are constants to be determined by history jobs. The integration on the above formula can be computed approximately choosing the interval 5 seconds.

4.2.1. Build the History Job Database Each history job is a record in the database. A record consists of two parts, the input part and the output part. The input part includes simulation configuration, load average of each application server, average expected completion times of running simulations on each application server. The output part includes actual completion time of the simulations on each application server.

To build the history job database, each simulation is executed on all application servers, with disabling of database constraints, until a satisfactory number of simulations are finished and stored in the database, for example, 200.

4.2.2. Compute Constants C_1 and C_2 The constants C_1 and C_2 can be computed by completed simulations. The average load average can be recorded during the execution of a simulation for each application server; the amount of data produced for each iteration can also be recorded. Therefore, the two constants can be computed by solving linear equation systems. Obviously, there are many more equations than parameters. To find reasonable constants C_1 and C_2, we can form the following mathematical problem. Let n be the number of records in the job history database. Let $x = (x_1, x_2)^t$ be the two constants to be solved. Let $A = (a_i j)n \times 2$ be the coefficients. Let $b = (b_1, ..., b_n)^t$ be the completion times for each job in the history database. Our goal is to minimize $\|b - Ax\|$.

We can apply the singular value decomposition (SVD) for A. Thus there exists a n by n orthogonal matrix U, a 2 by 2 orthogonal matrix V, and a n by 2 matrix Λ with the form $\Lambda = (diag(\lambda_1, \lambda_2), O)^t$ where λ_1, λ_2 are positive real numbers and O is a 2 by $n-2$ zero matrix, such that $A = U\Lambda V$. Let $y = Vx$ and $c = U^t b$, then $\|b - Ax\|^2 = \|b - U\Lambda V x\|^2 = \lambda_1^2 y_1^2 + \lambda_2^2 y_2^2 - 2c_1\lambda_1 y_1 - 2c_2\lambda_2 y_2 + c^t c$. Hence, when $y = (\frac{c_1}{\lambda_1}, \frac{c_2}{\lambda_2})^t$, $\|b - Ax\|$ is minimized. Therefore, we can compute x using $x = V^t y$.

4.2.3. Define Distance of Two Jobs For each record in the history job database, some attributes are numerical while others are categorical. To define the distance of two records, we can use the so called Heterogeneous Euclidean Metric. The distance of record x and y is defined as follows:

$$d(x, y) = \sqrt{\sum_i w_i * d_i(x, y)^2} \qquad (2)$$

The summation in the distance is taken over all attributes of the input part of the records. For categorical attribute i, the distance is 0 if they are same, 1 otherwise. For numerical attributes i, the distance is

$$d_i(x, y) = \frac{|x_i - y_i|}{\max x_i - \min x_i} \qquad (3)$$

w_i is the weight put on the attribute i, to identify more important attributes than others. For example, the expected

completion time of other running simulations has a huge impact for choosing which application server, thus we put more weight on this attribute.

4.2.4. Nearest Neighbor Algorithm For a new simulation, a nearest neighbor will be picked from the history job database. Then from the output part of this nearest neighbor, we can pick the application server which completes the nearest neighbor in shortest time. Then this application server will be the one to execute the new simulation.

The advantage of the new algorithm is to learn the execution patterns of simulations. It first predicts the completion time of the simulations for each application server, then assign the simulation to an application with the earliest completion time. It not only takes load average into account, but more attributes which may impact the completion time of simulations.

5. Simulation Resuming

As shown in Figure 1, simulation data and checkpointing data generated from the simulations running on application servers is stored in different databases, i.e., SD and SM. The two parts of data form one distributed transaction and this transaction spans two different database, therefore, we need to use the J2EE transaction service (JTS) to ensure data integrity. Data integrity is critical to implement the simulation-resuming.

The simulation jobs are normally long running tasks, for example, a normal simulation make takes several hours to accomplish. If some software or hardware failure occurs during the execution, we have to find ways to restart the simulation from a checkpoint prior to the time of the failure. Users may want to resume a simulation which was terminated earlier. In this case, we want to start the simulation from where the simulation was terminated.

To accomplish these cases, we need to checkpoint the simulation periodically during its execution. This raises several issues against the checkpointing procedure, for example, what information about the simulation should be checkpointed, how often the checkpoint should take place, and more importantly, how do we synchronize the simulation data and checkpoint data (that's why JTS and JTA involves). In the next few paragraphs, we'll provide answers to these questions.

5.1. What information should be checkpointed?

Basically, we want to checkpoint all information such that a simulation can be started using this information as inputs. Take the NOM project as an example, we want to checkpoint the following information:

1. Environmental variables, including requested simulation time and remaining time.
2. Molecule attributes, including counts of atoms, counts of function groups, molecule weights, etc.
3. Positions of molecules.
4. Reaction probability tables of all kinds of reactions for all molecules.

To restart a simulation, all the above data will be read by the simulation program. Each instance of molecule object will contribute one row to the checkpoint database table. When the number of molecules increases, the amount of data to be checkpointed also increase and so does the time required to finish the checkpoint process.

5.2. How often should checkpoint take place?

The frequency of checkpointing of a simulation depends on the quality of the simulation program. Based on our experience, some simulations may exhaust available physical memory and throws an OutOfMemoryErrorException during their execution. In this case, we want to checkpoint the simulations before they crash.

The frequency of checkpointing also depends on the requested execution time and the amount of data generated by the simulation. The simulation configuration determines statistically how many hours the simulation will execute and how much data will be generated. For example, if a simulation requires to run 24 hours, we may checkpoint it every 1 hour. If another simulation requires to run just 1 hour, we way choose not to checkpoint it, in the case of failure, we can start the simulation from the scratch.

The history of the archived simulations may also influence the frequency of checkpointing. This history data including user behavior (for example, a user might stop the simulation and resume it later), the frequency of premature termination of simulations. Data mining might be involved to determine the frequency of checkpointing.

Two basic approaches to determine checkpoint frequency are

- checkpoint interval: the number of megabytes of data generated by a simulation, for example, 10MB.
- checkpoint timeout: the number of minutes the simulation has been executing, for example, 30 minutes.

To further improve the reliability of simulations, at each checkpoint, an "I'm alive" message is inserted into a database with current timestamp. The job dispatcher described before will check the message to ensure that the simulation is still running.

```
import java.sql.*;
import javax.sql.*;
import oracle.jdbc.*;
import oracle.jdbc.pool.*;
import oracle.jdbc.xa.OracleXid;
import oracle.jdbc.xa.OracleXAException;
import oracle.jdbc.xa.client.*;
import javax.transaction.xa.*;

class Checkpoint{
  public static void main(String[] args)
      throws SQLException{
    try{
      //Create XADataSource instances
      OracleXADataSource oxds1=new OracleXADataSource();
      oxds1.setURL("jdbc:oracle:thin:@db1:1521:sd");
      oxds1.setUser("scott");
      oxds1.setPassword("tiger");
      OracleXADataSource oxds2 = new OracleXADataSource();
      oxds2.setURL("jdbc:oracle:thin:@db2:1521:sm");
      oxds2.setUser("scott");
      oxds2.setPassword("tiger");

      // Get a XA connection to the underlying data source
      XAConnection pc1  = oxds1.getXAConnection();
      XAConnection pc2  = oxds2.getXAConnection();

      // Get the Physical Connections
      Connection conn1 = pc1.getConnection();
      Connection conn2 = pc2.getConnection();

      // Get the XA Resources
      XAResource oxar1 = pc1.getXAResource();
      XAResource oxar2 = pc2.getXAResource();

      // Create the Xids With the Same Global Ids
      Xid xid1 = createXid(1);
      Xid xid2 = createXid(2);

      // Start the Resources
      oxar1.start (xid1, XAResource.TMNOFLAGS);
      oxar2.start (xid2, XAResource.TMNOFLAGS);

      // update SD and SM with conn1 and conn2
      updateSD (conn1);
      updateSM (conn2);

      // END both the branches -- THIS IS MUST
      oxar1.end(xid1, XAResource.TMSUCCESS);
      oxar2.end(xid2, XAResource.TMSUCCESS);

      // Prepare the RMs
      int prp1 =  oxar1.prepare (xid1);
      int prp2 =  oxar2.prepare (xid2);

      boolean do_commit = true;
      if (!((prp1 == XAResource.XA_OK) || (prp1 == XAResource.XA_RDONLY)))
          do_commit = false;
      if (!((prp2 == XAResource.XA_OK) || (prp2 == XAResource.XA_RDONLY)))
          do_commit = false;

      if (prp1 == XAResource.XA_OK)
        if (do_commit)
            oxar1.commit (xid1, false);
        else
            oxar1.rollback (xid1);

      if (prp2 == XAResource.XA_OK)
        if (do_commit)
            oxar2.commit (xid2, false);
        else
            oxar2.rollback (xid2);

      // Close connections
      conn1.close();
      conn1 = null;
      conn2.close();
      conn2 = null;
      pc1.close();
      pc1 = null;
      pc2.close();
      pc2 = null;
    }catch(SQLException e){
    }catch(XAException xe){
    }
  }
}
```

Figure 6. Checkpoint code snapshot

5.3. How to synchronized simulation data and checkpoint data?

JTS and JTA play an important role to synchronize the simulation data and checkpoint data. The two kinds of data will go to two different database in one distributed transaction. This transaction either commits or rollback and ensures the data in the two databases are integrated, using the X/open two phase commit protocol.

The following figure shows a snapshot of code to implement checkpointing.

5.4. Resume Simulations

To summarize, we employ JTS, JTA and data mining techniques to accomplish checkpointing of simulations. If a simulation does not respond the KEEPALIVE message in a timely fashion, it's marked as "crashed" by the dispatcher and thus needs to be restarted. The simulation will be restarted by reading the checkpoint data and thus resume from the checkpoint prior to the point of failure.

In the case of application server failure, the dispatcher can detect the failed application server through several ways. For example, the failed application server cannot upload its loadavg to the APPSERVER table, or simulations running on it cannot respond the KEEPALIVE message in a timely fashion. Once the dispatcher detects that the application server is down, all simulations running on it will be marked as crashed. They will be dispatched to other application servers by the dispatcher.

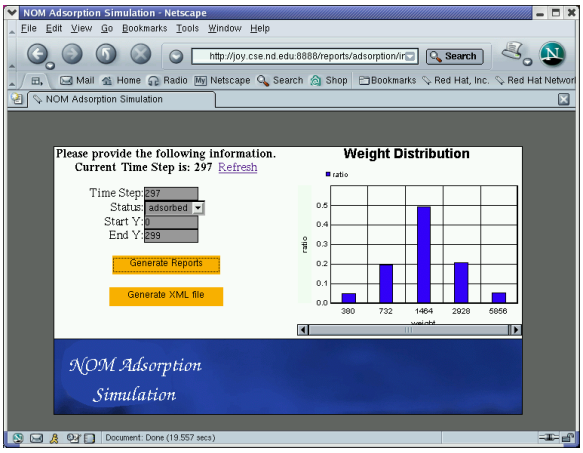

Figure 7. A sample graphical reports

6. Reports and Collaboration Tools

We provide two kinds of reports to the users through the Oracle Reports Server. The graphical reports of the data analysis can be delivered as Java Server Pages. The text reports can be delivered as XML files through XSQL (XML SQL) pages. The XML files can be transformed into HTML using XSLT (eXtensible Style Language Transform). Figure 7 shows a sample of graphical reports, which is generated dynamically using Oracle Reports.

To enable users to collaborate with each other, mechanisms are developed to let users share simulation configurations and results. For example, before the user submits a new simulation, all completed or currently running simulations with similar configurations are shown to the user. The user can view the reports for these simulations, instead of running the new simulation. Some other collaboration tools, such as discussion boards and chat rooms, are integrated into the architecture. Users can post articles and questions to the discussion boards and chat with other to discuss problems.

7. Conclusions and Future Work

We designed and implemented a multi-tier architecture to support a web-based environmental research portal. This architecture has the load-balancing and simulation-resuming features to achieve high scalability and reliability. This architecture can benefit scientists by providing a centralized research environment through which they can collaborate with each other.

More self maintenance features will be added to the web-based environmental research portal to improve its efficiency and reliability.

References

[1] R. A. A.C. Dusseau and D. Culler. Effective distributed scheduling of parallel workloads. In *Proceedings of ACM SIGMETRICS*, pages 25–36, 1996.

[2] S. Banawan and J. Zaborjan. Load sharing in heterogeneous queueing systems. In *Proceedings IEEE INFOCOM*, pages 731–739, 1989.

[3] P. Dinda. Online prediction of the running time of tasks. In *Cluster Computing, 5(3)*, 2002.

[4] E. L. D.L. Eager and J. Zahorjan. The limited performance benefits of migrating active processes for load sharing. In *Proceedings of ACM SIGMETRICS*, pages 63–92, 1998.

[5] M. Harchol-Balter and A. Downey. Exploiting process lifetime distributions for dynamic load balancing. In *Proceedings of ACM SIGMETRICS*, pages 13–24, 1996.

[6] W. Inmon. *Build the data warehouse*. John Wiley and Sons, New York, 1996.

[7] J2EE. *http://java.sun.com/j2ee*.

[8] W. Leland and T. Ott. Load balancing heuristics and process behavior. In *Proceedings of ACM SIGMETRICS*, pages 54–69, 1986.

[9] D. S. M. Rinard and M. Lam. Jade: A high-level machine-independent language for parallel computing. In *IEEE Computer, 26(6)*, pages 28–38, 1993.

[10] M. Mutka and M. Livny. The available capacity of a privately owned workstation environment. In *Performance Evaluation 12(4)*, pages 269–284, 1991.

[11] e. a. S. Kumara. Simulation anywhere any time: web-based simulation implementation for evaluating order-to-delivery systems and processes. *Proceedings of the 2002 Winter Simulation Conference*, pages 1251–1259, 2002.

[12] B. Siegell and P. Steenkiste. Automatic generation of parallel programs with dynamic load balancing. In *Proceedings of the Third International Symposium on High-Performance Distributed Computing*, pages 166–175, 1994.

[13] e. a. V. Holmes. Evolving the web-based distributed si/pdo architecture for high-performance visualization. *Proceedings of the 34th Simulation Symposium*, pages 151–158, 2001.

[14] W. Winston. Optimality of the shortest line discipline. In *SIAM J. Appl. Prob., 14:181-189*, 1977.

[15] J. M. X. Huang. Building a web-based federated simulation system with jini and xml. *Proceedings of the 34th Simulation Symposium*, pages 143–150, 2001.

[16] XML. *http://www.w3.org/XML*.