Large-Scale Simulation Experimentation and Analysis: Database Programming Using Java

Craig E. Ward and Ke-Thia Yao
Information Sciences Institute, Univ. of Southern California
4676 Admiralty Way, Suite 1001, Marina del Rey CA 90292
Phone: 310 822-1511 FAX: 310 823-6714 {cward,kyao}@isi.edu

ABSTRACT

High Performance Computing has made significant strides in the distributed simulation community. The Joint Forces Command has fielded more than a million independent agents in its JESPP project, with a concomitant data management challenge. Enabling and optimizing this transcontinental computing and analysis environment has drawn significant interest from the T&E community. This paper focuses on the Scalable Data Grid (SDG) project at the Information Sciences Institute and it illuminates why some Java techniques were found to be useful and some were not. The study of the programming will aid in examining the design and implementation of an effective distributed simulation database using the Java Programming Language and its associated tools and Application Programming Interfaces. The transition of intelligent agent simulations from training to experimentation requires the effectual logging, processing, storing, retrieving, and analyzing terabytes of data. The design, construction, and evaluation of the SDG strives to balance efficiency of execution, clarity of development, and security of the environment to create a robust, scalable system to support distributed simulation database population, organization, and utilization. The choice of the most appropriate programming language was central to the effective development and eventual utility of the SDG. The depth and breadth of Java technologies provide a rich set of capabilities. Not all of these capabilities are needed by all systems. The SDG was built using, among other things, Java Database Connectivity (JDBC) and Remote Method Invocation (RMI). The frameworks available for wrapping these and other technologies, such as Java2 Enterprise Edition ™ (J2EE), Java Data Objects (JDO) or Hibernate were not used. The rationale for these choices is laid out and reviewed. All of these choices are examined and the constructed classes are illustrated. The authors review lessons learned and performance evaluations across several dimensions. This paper covers design considerations that were made to effectively generate the SDG system code, including choice of language, programming tools, *etc.* All of these techniques should be extensible into the T&E community, as that community increases its use of HPC capabilities.

AUTHORS

Craig E. Ward is a Parallel Computer Systems Analyst at ISI/USC. He is active in programming on the Joint Experimentation on Scalable Parallel Processor project, focusing on the data management issues. He recently completed a master's thesis addressing the security issues related to the choice of a programming language. He worked for 20 years as a programmer, software engineer, and consultant. He has a B.A degree in History from the University of California, Irvine, and an M.S. in Computer Science from Loyola Marymount University.

Ke-Thia Yao is a research scientist in the Distributed Scalable Systems Division of ISI/USC. He is working on the JESPP project, which has the goal of supporting very large-scale distributed military simulation involving millions of entities. Within the JESPP project he is developing a suite of monitoring/logging/analysis tools to help users better understand computational and behavioral properties. He received his B.S. degree in EECS from UC Berkeley, and his M.S. and Ph.D. degrees in Computer Science from Rutgers University.

Large-Scale Simulation Experimentation and Analysis: Database Programming Using Java

1 Introduction

The Scalable Data Grid (SDG) is a system for distributed collection and analysis of large amounts of data.

Initial development and deployment of the SDG is as part of the Joint Semi-Autonomous Forces (JSAF) simulation environment of the Experimentation Directorate (J9) of the Joint Forces Command (JFCOM).

The JSAF simulations are being used in the *Urban Resolve 2015* experiments to analyze new ideas, methods, and tactics for solving combat problems in future hypothetical urban combat situations. The program includes networked workstations and machine clusters at locations across the United States. Workstations in Virginia, California, Kentucky, and elsewhere utilize 128-node clusters in Ohio and Hawaii. The clusters simulate the large numbers – the goal is 1 million entities – and varieties of entities found in urban settings. Military operations are simulated in the mist of this cultural milieu, generating multiple terabytes of data (Yao & Wagenbreth, 2005).

Figure 1. Map illustrating some of the networked sites of Urban Resolve 2015.

The SDG system collects data from the simulations, stores this data in tables in a relational database, and then uses this data to populate more tables that represent the data as multi-dimensional cubes. SDG plugs into the simulation environment as a replacement to a preexisting data logger. From the perspective of the simulations, SDG is the same data logger.

SDG augments the overall JSAF environment by extending the logging capabilities with enhanced analysis features. By generating cube views of the data, SDG enables online analysis processing (OLAP). The data may be quickly viewed in a “slice-and-dice” mode with different dimensions “rolled-up” or expanded as desired. SDG also limits the need to

move the massive amounts of raw data to a central location. As the number of entities increases the amount of data increases and soon it will not be feasible to move data to a central store (Yao & Wagenbreth, 2005). The SDG analyses distributed data on the local hosts and sends only relatively small summaries to the central site.

2 SDG Architecture

SDG implements an *n*-tier architecture. It is comprised of a set of Java applications, a web application built with Java servlets that run in a container, a backend database, and a thin client user interface utilizing Java Server Pages to create HTML. The user interface query functions use HTML Forms technology.

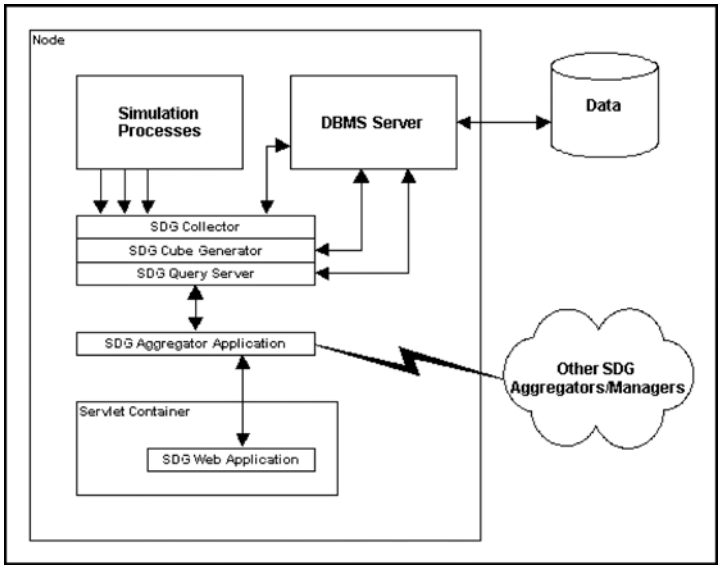

Figure 2. Basic architecture of SDG.

Configuration of the cubes to be created is setup using a GUI Java configuration tool. The tool allows an analyst to configure the roll-up criteria. A simulation entity can be categorized as a particular type. These types may themselves be placed into additional categories. A hierarchy of types is created for use in the cube generation stage of processing.

2.1 Java Applications

Four Java classes serve as the core of the SDG. One class acts as a data collector and presents the expected interface to the simulations. A second class uses the collected data to generate cube representations in the database. A third class handles queries for cube data and returns references to cube instances. The forth class is an aggregator that can receive remote cube objects from the query server and combine them into new cube objects. These may also be passed up a tree of aggregators to a top node that is used by the web application.

2.2 Configuration GUI

A GUI configuration tool is used to setup the descriptions within the database of the supported cubes.

The key output of the configuration tool is the partial orderings of the entities of the simulation. These partial orderings control how the cube generation rolls-up the measures for a cube.

2.3 Web Application

The SDG web application provides the core functionality from the user's perspective. Java servlets communicate with SDG manager objects. The servlets request cube definitions from the managers and convert the descriptions into objects easily displayed by Java Server Pages as HTML with JavaScript.

Queries are selected by the user from an HTML Form. As the user selects items in the form, JavaScript code manipulates the elements of the Document Object Model to change the options presented in the form. The user may elect to drill down to individual concrete entities for one dimension but allow another dimension to rollup to the most general classification. The JavaScript code allows the form to display just the options that are relevant for a particular dimension.

The data captured by the HTML Form is sent to a servlet using the HTTP POST command. The servlet converts the text data into a Cube View (described in more detail later) that the query server can then use to execute queries on the database.

The results of a query are returned to a servlet which processes the cube into Java objects which can be easily used by a JSP to generate a HTML display of the results.

2.3.1 Cube Views and Cube Descriptions

Cube Descriptions and Cube View perform different roles within the SDG. The description contains the set of possible coordinates, measures and their partial orders contained in a Cube. The descriptions are set before the simulations are executed. The view is a runtime selection of particular coordinates and measures that can be used to execute a query on the database so that an instance of a cube can be created consisting of just a selected subset of coordinates and measures.

2.4 Database Engine

The backend database engine is the relational database management system from MySQL. SDG communicates with the database server using the Java Database Connectivity (JDBC) driver supplied by MySQL.

3 Technologies Used

The implementation of SDG uses many of the latest features of Java Platform, Standard Edition 5 (Java SE 5). This section describes those technologies. A later section will discuss some of the reasons other features and packages for Java were not utilized.

3.1 JDBC

Interaction with the backend database is through the Java Database Connectivity package. SDG manages connections, cube generation, queries, reads, and updates directly.

The fundamental object persisted by SDG for analysis purposes is the *Cube Element*. An element is one or more measures located in a relational table based upon the coordinates for the measures in a cube. Each column in a row represents a coordinate or a measure.

The data logger inserts data into the database using SQL text statements. This conforms to the standard of the logger subsystem that SDG replaces. The text statements are submitted directly to the JDBC driver.

3.2 Distributed Communication

Communication between the SDG Cube Managers and aggregators uses Remote Method Invocation (RMI). This technology is well integrated into the Java runtime environment and is relatively efficient.

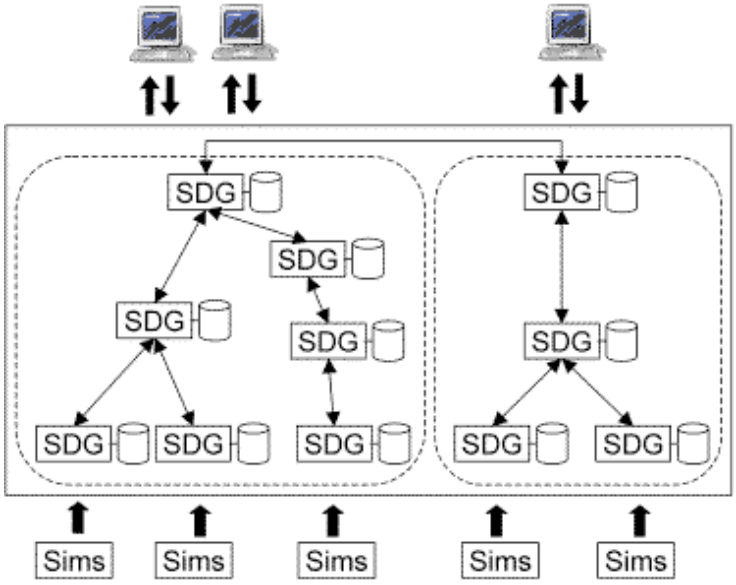

Figure 3: Distributed communications in SDG.

Two objects make up the vast majority of the RMI communication within SDG: *Cube Views* and *Cubes* with their *Paged Iterator*.

Cube Views are objects that describe the subset of a Cube that the user is interested in. The analyst selects entities and groupings of entities from a HTML Form. The resulting Cube View is serialized and passed to a Cube Manager/Aggregator. The manager may either use the Cube View to execute the query or it may pass the objects to child Cube Managers.

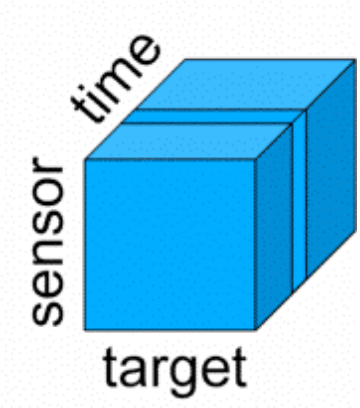

Figure 4: Example cube with three dimensions.

The result of a query is a Cube. More precisely, an instance of a Cube with the limitations set by the requested Cube View. The resulting cube object may be rather large, so the default approach to returning the data of a Cube is through a Paged Iterator.

A Paged Iterator is a RMI reference to a multi-threaded object that returns the elements of a cube in batches. The whole cube is not transferred as a single serialized object.

3.3 Threaded Processing

The core SDG application uses the new Java SE 5 thread capabilities as well as the traditional services. The three main control threads are submitted to an Executor service. Within each of these, more threads or services are created.

3.3.1 Client-Server Logger

The logger thread provides a TCP service for client simulations to use. It is this service which supports the previous data logger interface and allows SDG to transparently plug into the simulation environment. The service provides a text-based interface to the backend database server. Data is sent as SQL insert statements. The service also can be used interactively using a telnet client.

The implementation is based on the Java sockets package. As new requests are received, a new thread is created to service the request. Connections to the database are either created or reused from a pool of existing connection objects.

3.3.2 Cube Generator

The cube generator control thread uses an Executor service to run the three sub-threads of cube generation: creating messages, converting messages to facts, and using facts to update cubes.

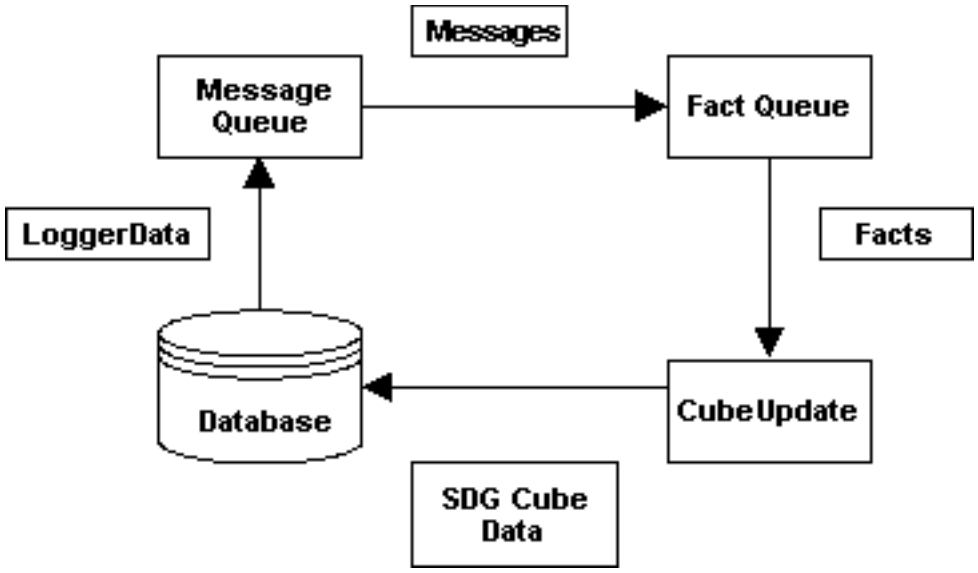

Figure 5. Diagram of cube generation process of SDG.

The first step in cube generation is the creation of a message from the data logged by the data logger. The message encapsulates the data needed to generate one fact for a cube. In the initial prototype, the message extracted the information from a Contact Report indicating which sensor initiated the contact, which entity was the target of the contact, and what the result was of the contact attempt.

The data for the messages can be spread across multiple columns or tables. Putting the knowledge of these locations into a message source isolates and simplifies the job of the fact generator.

Each message source provides a method that accesses a thread-safe queue, which is also a new feature of Java SE 5. The fact generator reads each message from the queue and processes the message data into a fact, which is then put into another thread-safe queue.

The key step in fact generation is the conversion of the entity identifiers from the logged simulation data into indices, or coordinates, for the cube. These are mapped from the data supplied prior to a simulation run with the configuration GUI.

The final step in the process involves reading a batch of facts from the fact queue and creating an in-memory instance of a cube based solely on that batch of facts. Facts with the same coordinates are added together. The cube is then filled-out by rolling up the facts based on the partial ordering configuration defined for the cube.

The in-memory instance of the expanded cube is processed one element at a time. Each element is used to either update an existing row or add a new row to the table representing the cube.

3.3.3 Query and Aggregation

The query engine thread accepts Cube View objects and returns an instance of the cube representing that view. Each SDG manager incorporates a factory class that can use a cube view to either execute a query on a local database or forward that view to one or more child managers for processing. Which technique is used depends on the startup configuration.

SDG Managers on leaf nodes interact with the local database using JDBC. SDG managers with child nodes communicate with them using RMI.

3.3.4 The Java Language and Security

By building SDG using the Java programming language, many of the security problems that plague other systems are not present. However, no language can shield the programmer from all vulnerabilities. The area where SDG security comes into play are from the sources of input text. Both the data logger and the web application process text from external sources.

The initial deployment environment for SDG is on a secure network. On such a network, the requirement that the SDG applications protect themselves from attacks is not part of the customer's requirements. With all of the physical access points secure, the possibility of malicious input is minimal. However, should SDG be equipped for deployment in less safe conditions, the objects that receive external input can be augmented without disrupting other classes in the system.

A particular attack that would need preventative measures is SQL Injection. In the case of the data logger, the conversion of SQL text into SDG-specific SQL statement objects would need to reject any unrecognized SQL text. The web application servlets would also need code to "white list" the acceptable text codes that come from the HTML Forms data.

3.4 Java SE 5

Implementation of SDG was greatly simplified by some of the new features found in Java SE 5. Many other technologies and features available to the SDG development team were available but not used. Some of these are addressed in the next section.

4 Technologies Not Used

The capabilities built into SDG are of themselves interesting. Also interesting, and perhaps of more general use, are the reasons behind the choices made while implementing the first versions of SDG.

4.1 Java EE

The Java Platform, Enterprise Edition (Java EE) is a set of technologies intended for distributed processing applications. It has features for persistence of objects to databases, resource naming, containers, and many other things. The problem with Java EE is that it is too much for most needs. It is a complex environment, difficult to use and easy to get wrong. (Tate and Gehtland, 2004) contains an excellent analysis of the complexities and

issues with Java EE. The paper (Gorton and Lui, 2003) presents an analysis of the performance characteristics of various Java EE implementations.

The SDG development team did, however, "cherry pick" from the Java EE tree. Java servlets and Java Server Pages are two of the simpler Java EE technologies. SDG uses the open source servlet and JSP container Tomcat. The role of these Java EE technologies is to provide a simple user interface to a lightweight, thin client web browser. Using more would make the system more complex than necessary.

The persistence needs of SDG are also simpler than the complexity of Java EE. The data logger is not persisting Java objects. SDG cube objects are not themselves persisted; only the individual elements, with their integer coordinates and simple associated measures stored in relational tables. Both Bean Managed Persistence and Container Managed Persistence require too much additional configuration and overhead to be of great use to SDG.[1]

4.2 Other Persistence Options

Java EE is not the only persistence product available. Others include Java Data Objects (JDO) and Hibernate. Each provides lightweight persistence of Java objects (PJOs – Plain Old Java Objects).

4.2.1 JDO

JDO is an object-oriented wrapper around JDBC. The intent is to free the developer from concerns over the plumbing of connections to a database and the nitty-gritty of relational tables. Java objects follow a protocol similar to that of a Java Bean. The code required to manage the persistence of an object is inserted into the byte-code created by the Java compiler. A persistence manager, provided by the JDO implementation, utilizes the additional code to manage reading, writing and updating the Java object. (Almer, 2002) provides a quick overview introduction to JDO.

According to (Baldwin, 2003), JDO can perform efficiently with large data sets. The analysis, however, did not include a large data set distributed across multiple systems.

4.2.2 Hibernate

The Hibernate persistence manager uses XML files to map the Java objects to the relational tables. The Hibernate API provides methods for getting stored objects from the database and methods for updating and creating new objects. The Java objects themselves do not have any code for dealing with databases. All of the work is done in the Hibernate configuration. A simple example project using Hibernate is in (Tate and Gehtland, 2004).

[1] Of course, technology is always changing. These statements reflect the authors views of the 2.0 and 2.1 versions of the Java EE specification. The specification may have simplified things since then. All such evaluations should be considered provisional.

The persistence needs of SDG, at least for the first operational deployment, are simple and sparse. However lightweight these systems, adding them where they are not needed increases complexity where the added complexity provides no added value.

4.2.3 Clustered JDBC

The open source project *Sequoia*, formerly Clustered JDBC (C-JDBC), is a JDBC driver capable of coordinating multiple, heterogeneous databases. C-JDBC supports load balancing , failover, and redundant operations. This is similar to what SDG is attempting, but it is not quite the same. The environment supported by the SDG does not require that every database contain the same set of tables. C-JDBC is used to present to applications a database cluster as a single, uniform database. For a more complete description, see (Cecchet, et. al. 2004).

4.3 Use what you need

Decisions regarding which technologies to use in implementing SDG were neither simple nor easy. Contractual rules mandated using free software. The developers used their collective experience and judgment to choose among available resources. Caution was required to avoid the use of a technology only because it was available and popular. Security, ease of implementation, and straight forward implementation of the design, were the critical factors.

5 Future Development

Experience with the initial deployment suggested several possible enhancements.

- The web application front end is sparse and could use better tools for display of multi-dimensional data. Technologies being evaluated include Java Server Faces (JSF) and Ajax.
- Caching of Cube View results.
- Develop an *ad hoc* query application.
- Implement an algebra for manipulating cubes.
- Monitoring the status of the distributed nodes to detect which one nodes are not performing as expected.
- Dynamic updates of cube descriptions based upon the messages sent from the simulations.
- Creating a new user interface that allowed for non-forms based interaction with the cube elements.
- Allow nodes to dynamically configure their communication links.

As technologies advance, and perhaps improve, the SDG development team will continue to evaluate and analyze choices to best adapt to new environments. The only constant is change.

6 References

Almaer, Dion. (2002). Using Java Data Objects. *ONJava.com* 2/6/2002. URL: http://www.onjava.com/lpt/a/1372

Baldwin, Richard T. (2003). Views, Objects, and Persistence for Accessing a High Volume Global Data Set. *Digest of Papers IEEE Symposium on Mass Storage Systems (MSS'03)* p 77-81

Cecchet, Emmanuel, Marguerite, Julie and Zwaenepoel Willy (2004). *C-JDBC: Flexible Database Clustering Middleware*. http://c-jdbc.objectweb.org/current/doc/C-JDBC_Flexible_Database_Clustering_Middleware.pdf

Gorton, Ian and Liu, Anna. (2003). Evaluating the Performance of EJB Components. *IEEE Internet Computing*, v 7, n 3, May/June, 2003, p 18-23

Tate, Bruce and Gehtland, Justin. 2004. *Better, Faster, Lighter Java*, O'Reilly Media Inc.

Yao, Ke-Thia, and Gene Wagenbreth (2005) Simulation Data Grid: Joint Experimentation Data Management and Analysis. *Interservice/Industry Training, Simulation and Education Conference (I/ITSEC),* 2005.