WebSphere. software

IBM® Trade Performance Benchmark Sample
for WebSphere® Application Server

IBM Trade Performance Benchmark Sample for WebSphere Application Server (otherwise known as Trade 6) is the fourth generation of the WebSphere end-to-end benchmark and performance sample application. The Trade benchmark is designed and developed to cover the significantly expanding programming model and performance technologies associated with WebSphere Application Server. This application provides a real-world workload, enabling performance research and verification test of the Java™ 2 Platform, Enterprise Edition (J2EE™) 1.4 implementation in WebSphere Application Server, including key performance components and features**.**

Overall, the Trade application is primarily used for performance research on a wide range of software components and platforms. This latest revision of Trade builds off of Trade 3, by moving from the J2EE 1.3 programming model to the J2EE 1.4 model that is supported by WebSphere Application Server V6.0. Trade 6 adds `DistributedMap`-based data caching in addition to the command bean caching that is used in Trade 3. Otherwise, the implementation and workflow of the Trade application remains unchanged.

Trade 6 also supports the recent DB2® V8.2 and Oracle® 10g databases. Trade's new design enables performance research on J2EE 1.4 including the new Enterprise JavaBeans™ (EJB™) 2.1 component architecture, message-driven beans, transactions (1-phase, 2-phase commit) and Web services (SOAP, WSDL, JAX-RPC, enterprise Web services). Trade 6 also drives key WebSphere Application Server performance components such as dynamic caching, WebSphere Edge Server, and EJB caching.

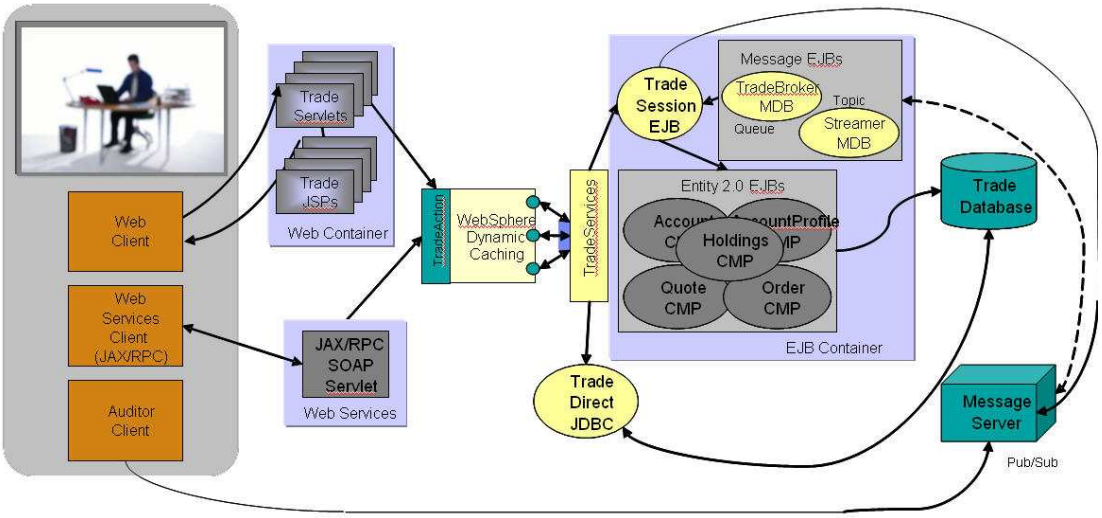

Figure 1: Trade 6 topology overview

This document provides an architectural overview of the Trade 6 performance sample package and design.

Installation overview and supporting database requirements

The installation process for Trade 6 is provided in the `Readme.html` file that is located within the installation package. This installation process involves the following four steps:

1. **Prepare for installation** – Download and unzip the installation package, setup the appropriate environment variables for DB2 if necessary, and start the target server or deployment manager for cluster installations.
2. **Create the Trade database** – Create the Trade database and associated tables on the desired database system.
3. **Run the configuration and installation scripts** – Using the `wsadmin` utility, execute the `trade.jacl` script that is provided in the installation package. After prompting for information about the database and application server configuration, this script configures the supporting resources within WebSphere Application Server and installs the Trade application.
4. **Populate the Trade database** – Click (Re)-populate Trade Database which is located in the configuration section of the Trade application. After this step is completed, you are ready to start using Trade.

This installation process is very similar to that used with Trade 3; however, the Jacl scripts that are used to configure the WebSphere Application Server resources and to install the application are streamlined to better support single server and cluster configurations. Furthermore, additional functionality is built into the scripts to support silent installations.

As previously mentioned, Trade 6 is specifically designed for J2EE 1.4 and WebSphere Application Server 6.0. Trade 6 also places additional requirements on the supporting database software that is used to store the Trade application data. To deploy and run the Trade 6 application, one of the following databases must be used:

- DB2 Universal Database V8.2
- DB2 Universal Database for z/OS Version 8
- Oracle 10g Database

Note: Trade 6 for z/OS is only supported in environments where DB2 Universal Database and WebSphere Application Server reside on the same system. The DB2 Universal JDBC Provider is configured in Type-2, Non-XA mode to take advantage of

performance benefits associated with the z/OS Resource Recovery Service (RRS). Furthermore, Trade 6 installations on z/OS platforms are currently limited to single server configurations.

Trade architecture overview

The Trade package provides a suite of IBM-developed workloads for characterizing performance of Java 2 Platform, Enterprise Edition (J2EE) application servers. The Trade application models an electronic stock brokerage providing Web and Web services-based online securities trading. Trade provides a real-world, business application mix of transactional enterprise beans, message-driven beans, servlets, JavaServer Pages™ (JSP™) files, Java database connectivity (JDBC™) and Java Message Service (JMS) data access, adjustable to emulate various work environments.

Figure 1 shows the high-level Trade application components and model-view-controller topology. The following sample screen-shot illustrates a user's home page after login. The home page details the account information and current market summary information.

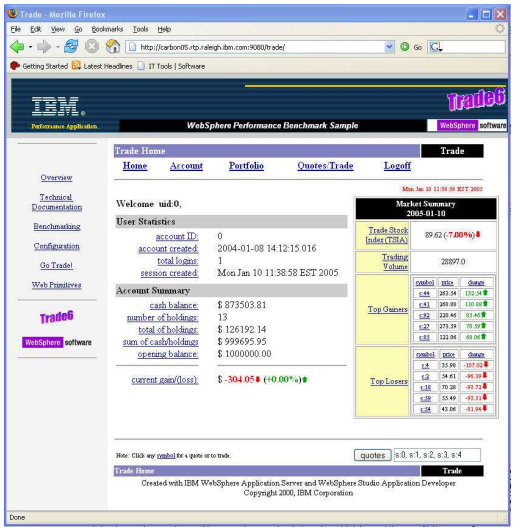

Figure 2: Trade Login home page

J2EE 1.4 performance research

EJB 2.1

Trade 6 continues to use the following features of EJB 2.0 and leverages EJB 2.1 features such as enhanced Enterprise JavaBeans Query Language (EJBQL), enterprise Web services and messaging destinations.

- **Container-Managed Relationships** – One-to-one, one-to-many, and many-to-many object to relational data managed by the EJB container and defined by an

abstract persistence schema. This feature provides an extended, real-world data model with foreign key relationships, cascaded updates and deletes, and so on.

- **EJBQL –** Standardized, portable query language for EJB finder and select methods with container-managed persistence.
- **Local and Remote Interfaces** - Optimized local interfaces providing pass-by reference objects and reduced security overhead.

WebSphere Application Server provides significant features to optimize the performance of EJB 2.1 workloads. Trade uses access intent optimization to ensure data integrity while supporting the highest performing and scalable implementation. Using access intent optimizations, entity bean run-time data access characteristics can be configured to improve database access efficiency which includes access type, concurrency control, read-ahead, collection scope, and so forth

Messaging

One of the most significant additions to the J2EE 1.3 programming model required support for transactional messaging through the JMS API. This included a new EJB component type, message-driven beans (MDB). MDBs are asynchronous, transactional message consumers. Trade integrates both queue-based and publish/subscribe MDBs for performance research and validation. In J2EE 1.4, the APIs for accessing queues and publish/subscribe resources are combined into one class hierarchy. Trade 6 is modified to use this single class hierarchy. This following diagram and associated description of ordered events demonstrate how the TradeBroker MDB is used in the Trade application to process purchase orders.

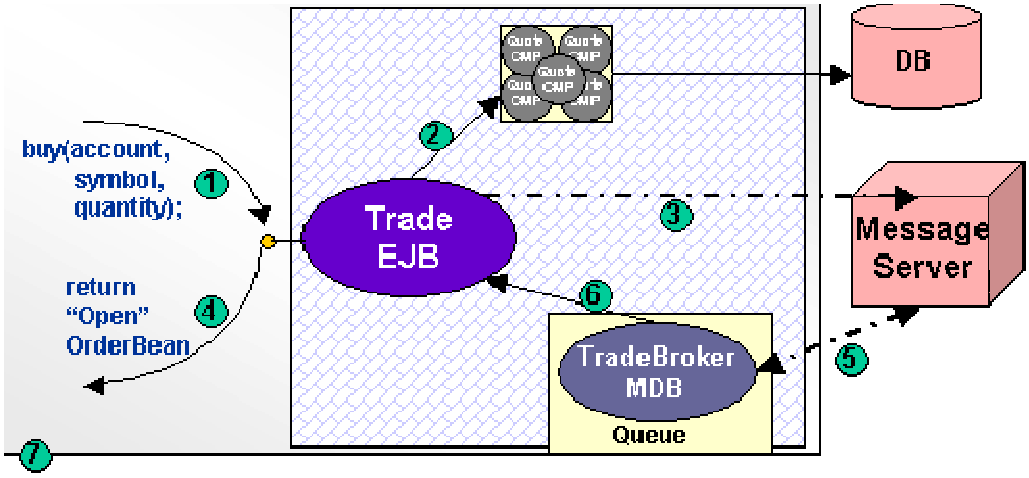

Figure 3: Trade asynchronous processing with MDBs

1. The purchase order is submitted by the user.
2. A new, “open” order is created in database.
3. The order is then queued for processing on the TradeBroker queue.
4. Creation of the “open” order is confirmed to the user.
5. The message server delivers the new order message to the TradeBroker MDB consumer.
6. The TradeBroker MDB processes the order asynchronously, completing the purchase.
7. The user receives confirmation of the completed order on a subsequent request.

The pub/sub based QuoteStreamer MDB subscribes to a `QuotePriceChange` message topic. Quote data for individual stocks is published to this topic as prices change, providing a real-time streaming quote feature in Trade. This streaming quote data is visualized using the Trade Streamer client.

Transactions

Operations spanning database and message resources such as the previous purchase order require an XA 2-phase commit to ensure consistency. Trade provides three order-processing run-time modes for transactional performance research. These modes include synchronous 1-phase, asynchronous 1-phase, and asynchronous 2-phase. By default, Trade starts in synchronous 1-phase mode. However, the order-processing mode can be selected dynamically during run-time for benchmarking. To simulate true XA 2-phase operations, use the asynchronous 2-phase mode in the configuration options.

Direct

The J2EE programming model provides managed, object-based EJB components. The EJB container provides declarative services for these components such as persistence, transactions, and security. The J2EE programming model also supports low-level APIs such as JDBC and JMS. These APIs provide direct access to resource managers such as database and message servers.

Trade provides a Direct implementation of the server-side trading services using direct JDBC. This implementation provides a comparison point to container-managed services that details the performance overhead and opportunity associated with the EJB container implementation in WebSphere Application Server.

JAX-RPC and enterprise Web services

All server-side trading services are provided as Web services in Trade. These services provide a workload to drive the Web services support in WebSphere Application Server. In addition to the EJB and Direct run-time modes, a Web Services run-time mode is provided. In this mode, the Trade Web application accesses the server-side services through the Web services (SOAP) protocol leveraging the Trade WSDL. A stand alone Web services client is also provided for Trade as a sample.

WebSphere Application Server performance components

Dynamic caching

WebSphere Application Server V6.0 provides two new interfaces, `DistributedObjectCache` and `DistributedMap`) by which J2EE applications can cache and share Java objects by storing a reference to the object in the cache. These interfaces expand on the command bean and servlet fragment caching capabilities already provided in previous versions of WebSphere Application Server.

Trade is designed as a performance sample application for the WebSphere dynamic caching service, providing not only a basis for performance research, but also a sample for configuring customer applications. By default, the caching capabilities within Trade are disabled. However, the caching type can be modified during run-time to use either the `DistributedMap` interface or command bean caching.

Furthermore, servlet and JSP fragment caching can be enabled through the administrative console. If servlet and JSP fragment caching is enabled, any Edge Side Includes (ESI) enabled Web server in front of the application server will also cache ESI fragments.

A cache hit is generally served in a fraction of the time of a normal request. Significant performance gains of 2x or more are generally achieved with Trade when leveraging the dynamic caching technology.

WebSphere Edge Server

Dynamic caching also has the ability to populate and control external caches on Web servers such as the Websphere Edge Server. Edge Server scenarios include multi-tier, clustered, demilitarized zone (DMZ), secure and other topologies providing unique performance benefits for a wide range of enterprise requirements.

Trade benchmarking

Typically, a complex series of scripts for a workload generation tool such as Mercury LoadRunner™ or WebSphere Workload Simulator are required to drive the various operations within Trade. However, Trade provides a scenario-driven workload to simplify the benchmarking process. In either case, the load generation tool selected to drive the Trade workload must provide cookie support to handle HTTP sessions.

A single URL can be driven under load with multiple client threads to completely cover the operations available within Trade. The TradeScenario servlet is responsible for emulating a typical user performing standard trading operations such as login, buy, get quote, view portfolio, and so forth. Driving the scenario servlet with multiple client threads models a population of users hitting the Web site during a daily trading session.

Using the TradeScenario servlet to drive the workload does have a drawback when compared to using a series of more complicated load scripts. The scenario servlet is responsible for managing clients and emulating user operations by dispatching simple client requests to complex Trade actions. The application server spends a large percentage of time performing work that is typically handled by a client or a more complex load driver. Consequently, performance numbers are artificially deflated when using TradeScenario servlet as compared to driving the Trade workload directly.

Trade also provides various runtime options providing performance comparisons for a set of runtime modes and configurations. These options are detailed in the Trade configuration page and include options for runtime mode such as EJB, Direct and Web Services. Another example is order processing mode which selects between asynchronous and synchronous order processing with a 1 or 2-phase commit.

Trade also provides a full suite of primitive tests to drive individual components of the WebSphere programming model. These include simple tests to drive servlets, JSPs and JDBC access as well as more complex primitive tests for EJBs and MDBs.

Trade development

Trade 3 was developed from the ground up using the WebSphere development tool WebSphere Application Developer (WSAD). Trade 6 was developed with the Rational Application Developer (RAD). RAD is built on the open source Eclipse™ technology that is found at http://www.eclipse.org

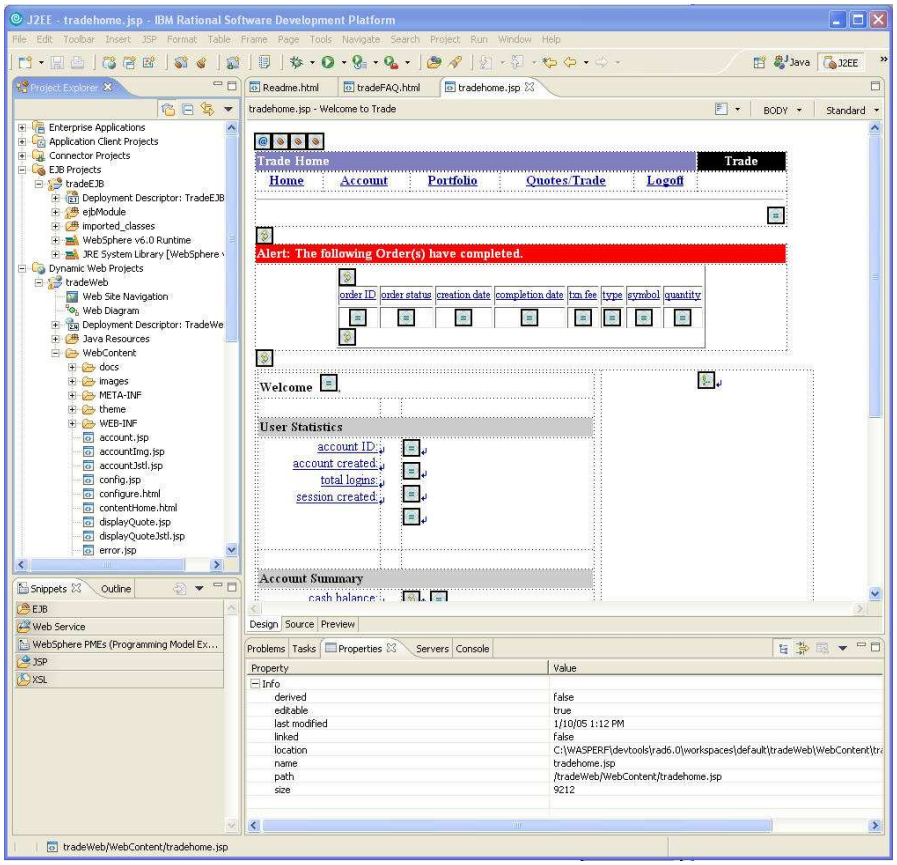

Figure 4: Trade Development in Rational Application Developer