The Propel Distributed Services Platform

Mike Carey, Steve Kirsch, Mary Roth, Bert Van der Linden, Nicolas Adiba, Michael Blow, Dana Florescu, David Li, Ivan Oprencak, Rajendra Panwar, Runping Qi, David Rieber, John Shafer, Brian Sterling, Tolga Urhan, Brian Vickery, Dan Wineman, and Kuan Yee

Propel
1010 Rincon Circle
San Jose, CA 95131
USA
carey@propel.com

Abstract

The Propel Distributed Services Platform (PDSP) is the core software product of Propel, a new Internet infrastructure software company. The PDSP product was created to enable Java developers to architect, implement, deploy, and maintain Internet applications and services much more quickly and easily than before while still providing all of the RAS (reliability, availability and scalability) that such applications require. In this presentation, we provide a brief overview of PDSP's key features, including its support for reliable and scalable data management, text indexing and searching, and persistent queuing. We also discuss its integrated Java APIs, built-in support for online-deployable data and schema changes, and system administration facilities.

1. Introduction

In the past, the development and maintenance of mission-critical e-business applications and services has proven to be a daunting task. Developing such an application has required piecing together a number of independent components, each with its own API, into a functioning multi-tier architecture that solves the given problem. A typical application might require the use of such components as a relational DBMS, a search engine, a queuing (or messaging) package, a directory server, an enterprise integration package, a system management package, and so on. Architecturally, the solution usually consists of a tier of web servers for handling connections and serving static content, a tier of application servers for running the application's business logic, and a third tier consisting of the aforementioned backend components. There are a number of significant problems with this approach:

- Building an application in this manner requires dealing with a number of disparate APIs provided by the different component vendors.
- Logic that spans components, such as a query to find in-progress auctions under $750 for items containing the keywords "Fender", "bass", and "guitar", requires hand-coding, hand-optimizing, and hand-integrating the results of queries against multiple APIs.
- Transactions that span components, such as a checkout process that locally records a new order and queues an interaction with the legacy order management system, can easily result in expensive two-phase commit coordination among different components.
- Managing state information such as user sessions in a failure-tolerant way requires the utilization of proprietary features provided by the particular application server being targeted for deploying the solution [1].
- Managing the resulting software system is complex, requiring individual configuration and management of all of the aforementioned solution components.
- Scaling the system to handle additional load is also complex; adding web servers and (to a lesser extent) application servers is generally feasible, but bottlenecks can appear on the third tier, in which case scaling becomes component-dependent and can be difficult to achieve, at least without buying expensive SMP hardware.

Proceedings of the 27th VLDB Conference,
Roma, Italy, 2001

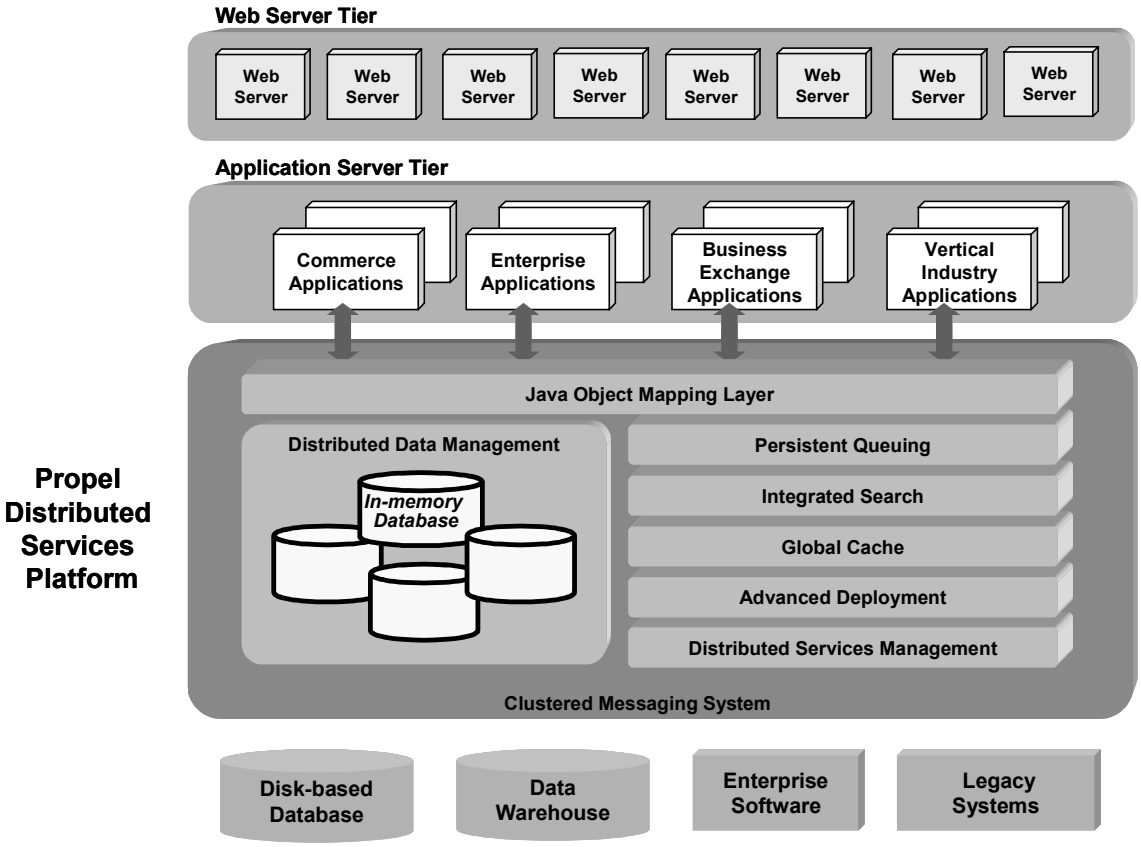

Figure 1: Propel Distributed Services Platform

2. The Propel Distributed Services Platform

The Propel Distributed Services Platform was created to overcome these problems, providing Java developers with a much stronger foundation upon which to build mission-critical Internet applications. In particular, PDSP aims to significantly simplify the development of *data-centric* Internet applications, i.e., applications that are centered around a data model and a database, especially those that require significant levels of availability and scalability. To provide the desired foundation, PDSP provides a number of important services that are pre-integrated for use by application developers. As indicated in Figure 1, these services sit in the gap between the application server tier and the third (or backend data) tier.

2.1 Clustered In-Memory Data Management

To provide fast access to e-business data, PDSP provides a unique, clustered in-memory data management service. PDSP's data management architecture was designed from the outset to be *incrementally scalable*, providing a way to scale the data management portion of an application. This offloads the backend disk-based database, thereby preventing application scalability problems due to the backend DBMS becoming a bottleneck.

The PDSP data management architecture is based on the time-proven *shared-nothing* architecture for scalable database systems [2]. In PDSP's case, the shared-nothing system is made up of a collection of in-memory database systems – TimesTen [3] is used internally as the per-node storage and query engine in PDSP – plus a disk-based database system such as Oracle or DB2. As Figure 2 indicates, PDSP keeps track of how the tables in the database are laid out physically within the architecture and utilizes distributed query processing techniques to provide applications with a location-transparent (i.e., single-image) view of all their data.

Tables managed by the clustered in-memory data management layer are replicated one or more times, as specified by the DBA. For read-only tables, multi-master replication can be used to provide scalability as well as availability for tables that fit within a single TimesTen node. For read-write tables, master-slave replication can be used to provide the desired level of availability. Data partitioning can be used to scale larger tables and/or (when combined with replication of partitions) to support scalability in the face of updates.

2.2 Integrated Text Search

Because many e-business applications require the ability to query database tables through both text searching (e.g.,

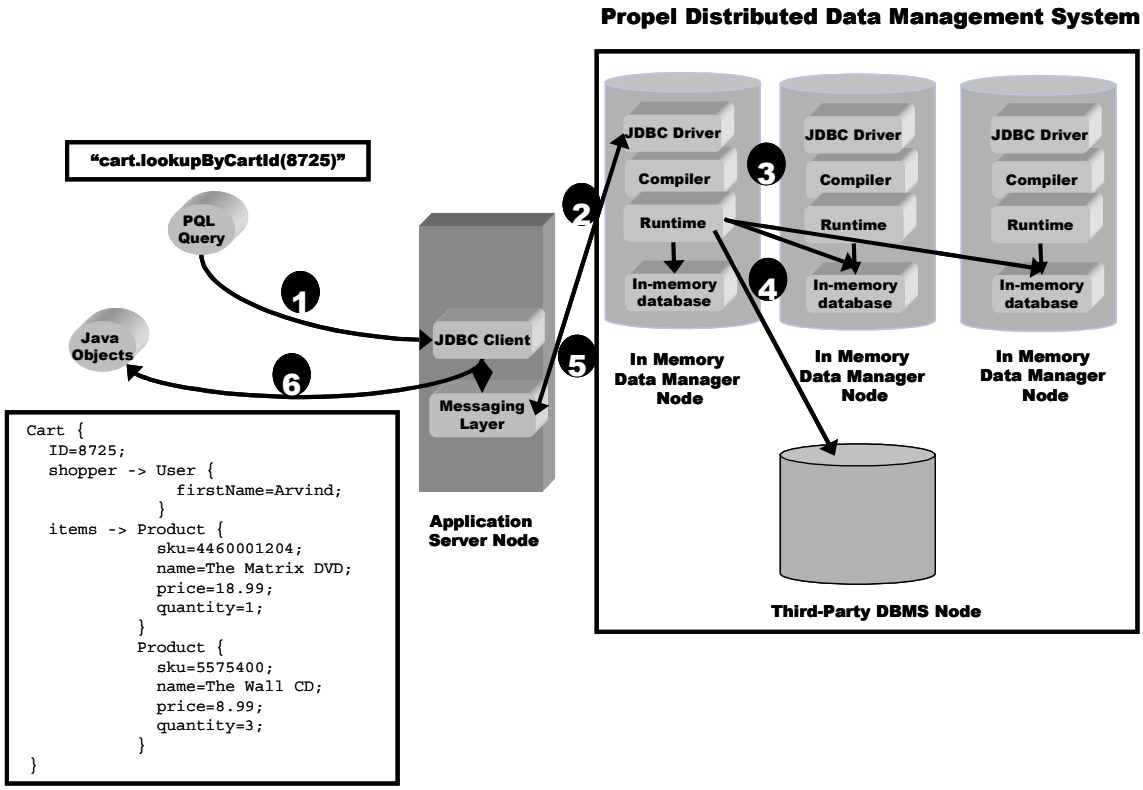

Figure 2: Propel Distributed Query Processing

of auction item descriptions) and parametric querying (e.g., of item prices) – often simultaneously – PDSP provides an integrated text search capability within its clustered, in-memory data management service. Text attributes can be indexed using either traditional database indexes or inverted text indexes, and in the latter case, queries can involve keyword-matching predicates as well as regular parametric predicates. Queries involving keyword-matching predicates can return relevance-ranked search results.

Once created, text indexes are automatically managed and exploited for query processing purposes by the same clustered in-memory data management infrastructure that was just described for record-oriented data. When tables with text indexes are placed and replicated for scalability, their indexes "go along for the ride"; as a result, queries involving text predicates scale up in the same manner as other queries. PSDP uses TimesTen tables and indexes internally to implement its search capabilities, as joins are much less costly in an in-memory DBMS than they would be in a traditional disk-based DBMS (making it possible to take a database-based approach).

2.3 Persistent Queuing

Another common requirement for e-business applications is the ability for transactional application components to interact asynchronously yet reliably. In the interest of providing "everything necessary" to easily build and maintain such applications, PDSP's clustered in-memory data management service also includes integrated support for persistent queues. Persistent queues are like "active tables" that have schemas determined by the application developer. Entries can be enqueued (rather than inserted) by one transaction and dequeued (rather than selected) by another. Dequeue operations specify predicates indicating the properties of the entries that they wish to dequeue, and they block rather than returning in the case when the queue currently has no matching entries. Two variants of dequeue are supported, one that removes the dequeued entries and another that simply updates them in an application-specified manner. Internally, PDSP uses TimesTen tables to implement its queues; as with regular PDSP data tables, replication is supported (and strongly recommended) to ensure high availability.

2.4 Java Object APIs

As indicated on the left-hand side of Figure 2, PDSP application developers interact with application data via Java objects rather than having to program using JDBC and SQL. PDSP provides an API layer called the Java Object Mapping (or OM, for short) layer. The OM layer provides an object/relational mapping facility that provides convenient JavaBean-based APIs for use in interacting with the application's data, including its record-based data as well as any text attributes and persistent queues.

The JavaBeans used in an application are specified via an XML mapping specification and then auto-generated by the OM layer's mapping tool. Flexible relationship mappings are supported, including binary 1:1, 1:N, and M:N relationships as well as general *N*-ary relationships. Relationships can have attributes, and relationships are modeled as interconnected Java objects or collections thereof. Support is provided for Java class hierarchies; rows in tables can be mapped to instances of classes in a hierarchy based on the runtime value of a discriminator attribute. The OM layer provides a facility for specifying query-based lookup methods a priori at bean generation time; the class of supported query predicates is quite rich, including support for path expressions, text predicates, and a number of other advanced features. Construction of ad hoc queries at runtime is also supported.

2.5 Deployment Support

A distinguishing feature of PDSP is the infrastructure that it provides for deploying application changes online (e.g., for deploying a new product catalog with no site downtime). The OM layer provides built-in support for versioned tables – tables in which a given object can have a (linear) sequence of versions associated with it – which enables application developers to write their Java code without regard for versioning. At runtime, version selection is performed automatically via the use of additional predicates emitted by the OM layer's mapping step; the version shown to a given application is determined by accessing its session information, and several versions of a row may co-exist and be accessed simultaneously. Updates to rows of versioned tables create new row versions rather than updating the existing versions in place. Data rows and the OM mapping information are both versioned in order to support online deployment of data as well as schema changes. The OM layer also handles the enforcement of referential integrity for versioned tables.

2.6 Global Cache Service

In addition to providing scalable data management services, PDSP provides a global cache service that provides an additional dimension of scalability by permitting commonly-accessed, dynamically-generated content to be cached and reused rather than having to be regenerated from the Platform data management service on each use. PDSP's caching architecture differs from traditional application server caches due to its global nature – rather than simply placing a cache on each application server, PDSP provides this type of caching but then also includes an additional layer of caching via a global cache server – data moves between the local caches (L1) on the application servers and the global cache server (L2) based on LRU management. The global cache service is primarily intended for use as a page fragment cache with TTL-based consistency management.

2.7 Distributed Services Management Console

The Propel Distributed Services Platform was designed to run on networks made up of many small (but memory-rich) inexpensive servers. To make the system and its applications manageable, PDSP has a centralized, web-based system administration console called the Propel Distributed Services Manager. This web-based console allows administrators to centrally configure and monitor the state as well as the performance of even large PDSP applications

3. Simplifying RAS for PDSP Applications

As mentioned earlier, the Propel Distributed Services Platform is designed to make it much easier for Java developers to create, deploy, and maintain data-centric, high-RAS, Internet applications [4]. This is accomplished in several key ways.

Internally, PDSP is built on a clustered messaging system that provides IPC for the Platform's various services and processes. The Propel clustered messaging system is itself a scalable, reliable, distributed service; it is cluster-aware and provides failure detection/handling and load balancing for all components of PDSP.

At the data management level, PDSP supports the data replication and partitioning options mentioned earlier. The provision of these options allows the DBA to select application-appropriate levels of both availability and load-handling capacity.

The inclusion of database queues in PDSP provides still another dimension to application scalability. In particular, PDSP's persistent queuing support enables applications to be constructed as a set of individually scalable services that consume and produce queue entries.

References

1. A. Rana et al, "E-business Developer: Prevent Clusterphobia", *Intelligent Enterprise,* Vol. 4, No. 8, May 24, 2001.

2. D. DeWitt and J. Gray, "Parallel Database Systems: The Future of High Performance Database Systems", *Communications of the ACM*, Vol. 35, No. 6, June 1992.

3. TimesTen Performance Software, "Architected for Real-Time Data Management: TimesTen's Core In-Memory Database Technology", *White Paper,* April 2001.

4. M. Carey et al, "Towards a Scalable Infrastructure for Advanced E-Services", *IEEE Data Engineering Bulletin*, Vol. 24, No. 1, March 2001.