WHITE PAPER

Progress ObjectStore

REAL-TIME DATA CACHING FOR ENTERPRISE APPLICATIONS

www.progress.com/objectstore

PROGRESS SOFTWARE

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This white paper explores the role that data caching technology can play in providing real-time data access for performance-critical, distributed applications. As part of that discussion it introduces Progress® ObjectStore®, a richly-featured data caching solution that combines the speed of in-memory data access with the transactional integrity of a persistent data cache—thereby providing enterprise applications with the best of both worlds.

The proven Cache-Forward™ Architecture (CFA) of ObjectStore complements existing corporate databases to increase the performance and scalability of transaction-intensive applications. Traditional relational databases serve a vital function within enterprise architectures, but they are under siege by modern distributed applications and the rising demand for real-time data access. With ObjectStore, application architects can eliminate the data bottlenecks that often constrain the performance of new distributed applications.

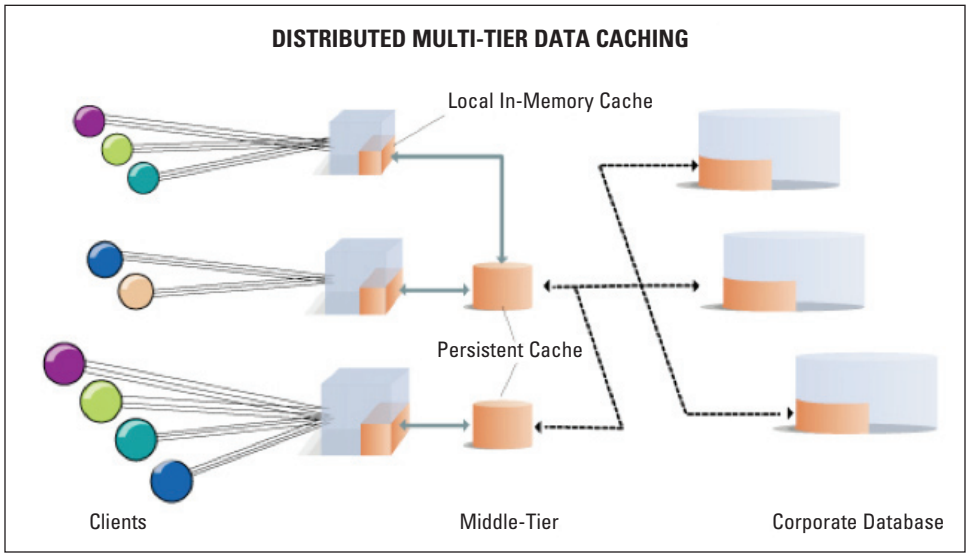

Figure 1: ObjectStore eliminates data bottlenecks, extending access to edge of enterprise

ObjectStore provides a high speed, data caching front-end for multi-tiered architectures, with near-real-time access and complete data integrity for transaction-intensive applications. It augments the corporate database of record, extending its utility to support distributed applications and their high volume requirements. ObjectStore technology has successfully delivered a rich array of data caching services to such prominent customers as Starwoods, Orange UK, Nomura International, and Delta Airlines.[1]

1. Additional information about ObjectStore and other data integration products can be found at www.progress.com/objectstore

THE DEMAND FOR DATA CACHING

The strategic initiatives of modern organizations continue to make tremendous demands upon their computing infrastructure. Business systems must accommodate both greater numbers and different types of users in support of an ever-expanding set of business applications. Many IT organizations have evaluated these challenges and looked to the flexibility and scalability of distributed architectures as a way to meet them. Distributed computing has become the foundation for solutions that can address requirements for Web/mobile access, online transaction processing, inter/intra-company business processes, and a range of new business models.

Distributed Computing – Impetus for Data Caching

Distributed computing, including SOA, deconstructs monolithic systems into a network of information services, requiring all the services to work together for the overall solution to be effective. Essential to this distributed model is flexible, scalable and performant access to relevant data for application coherence. The manner in which the middle tier is implemented will vary from deployment to deployment—regardless of the method by which the functionality is delivered, distributed architectures require a rich array of features and a robust transaction capacity. Without such a foundation, many of the vaunted efforts at new solutions are bound to falter.

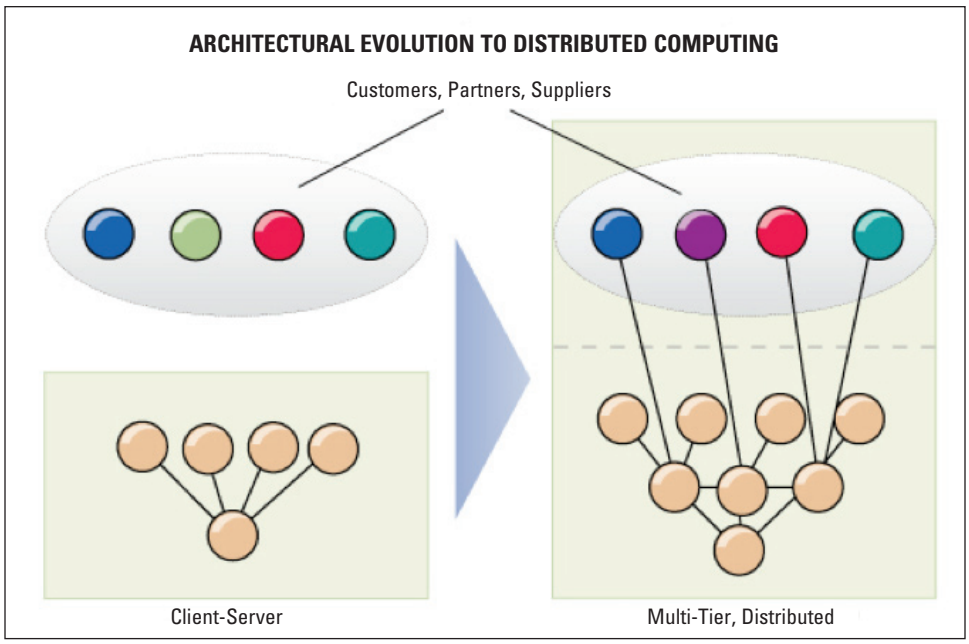

Figure 2: Architectures are evolving into more open, inclusive distributed computing models

But system architects and application designers often find that their architectural "reach" exceeds their solution delivery "grasp." Though there is enormous potential in the distributed model, execution has proven no easy task. Many solutions struggle to meet expectations because architectural design has not properly appraised the overall demands that distributed computing places upon each component of the end solution.

Planning for Performance

One notable area is system performance. Poorly implemented distributed architectures often create performance bottlenecks that seriously undermine the strategic goals of the organization. These performance deficiencies are not hidden in the dungeons of IT; they are exposed to the outside world. Many distributed systems are the "public face" of the organization to its customers, partners, suppliers, and other constituencies. When a bank gives its customers an online system, but provides achingly slow response times to its users, the bank suffers the consequences—in terms of customer dissatisfaction and possible defection to competitors. When a product wholesaler creates an online catalog for distributors and retailers that lacks accurate, up-to-the-minute inventory and pricing information, the core business model is undermined. In each instance, poor execution of a distributed architecture may end up driving customers to the competition. That's not an IT problem—that's a bottom line problem.

Properly calibrating system performance is critical. In distributed computing, performance is a feature not an afterthought. System architects must ensure that system performance measures up to the business goals of the organization. That requires performance to be incorporated as a fundamental part of system design—on a par with functional design, product features, and user interface. And one cannot simply look at more hardware as the answer to the performance problem. Distributed architectures have changed the arithmetic of computing—more hardware will not automatically deliver more capacity or sustainable performance improvement.

Exacerbating the Performance Challenge—The Move to Real-Time

The challenges of distributed computing are compounded by demands to accommodate growing transaction rates while maintaining—or improving—system responsiveness. Failure to fully understand the implications of these twin imperatives and appropriately address them can have dramatic impact. Examples abound:

- Conventional order entry applications—often designed for a small number of data entry clerks—have been subsumed into online "demand chain" applications that must service hundreds or thousands of simultaneous users who now order products online.

- Location based marketing, services and media—Using a Geographical Information System (GIS) to process consumer profiles and location with company resources, assets and logistics affords location based offerings. Companies may deliver relevant information and offers in real time through portable devices based on current geography.
- Traditional customer service applications could queue up incoming calls and ask customers to wait for the next available representative. Now Web-based self-service can by-pass the agents—eliminating that bottleneck, but also exposing performance deficiencies in now often-overwhelmed back-end systems.
- In financial services, stock traders no longer have to wait for the daily newspaper to get a stock quote, nor for a broker's office hours to trade securities. On-line trading and portfolio management applications now allow you to trade stocks from your computer—with hundreds, if not thousands, of different investment vehicles.

In each instance—and many others—the volume of transactions is exponentially greater, but there's been no easing of the demand for system responsiveness. In fact, access to distributed systems by users outside the enterprise virtually guarantees even *more* burdensome requirements for system responsiveness. Internal users may have little recourse if performance is substandard; they can be conditioned to accept the situation. But external users like customers have choices; competition is that proverbial "click away."

The Data Access Problem

Increased transaction volume, in tandem with the demand for real-time, up-to-the-minute accuracy—shines a spotlight on a critical challenge. Modern distributed solutions are often data-constrained. The responsiveness of traditional database architectures lags behind the high volume transaction requirements of new distributed applications.

Database management has long has been the province of products like Oracle, Sybase, Informix, and DB2. These relational architectures have a proven ability to support traditional enterprise solutions. They are the foundation for enterprise resource planning (ERP), customer relationship management (CRM), and other line-of-business (LOB) applications. But the structure of a relational database can be an encumbrance to real-time access. Relational databases depict a business entity (e.g. a customer, as shown in the accompanying figure) as a series of table entries. And database normalization advocates a multiplicity of tables to ensure data integrity, providing a single point of change for any one value.

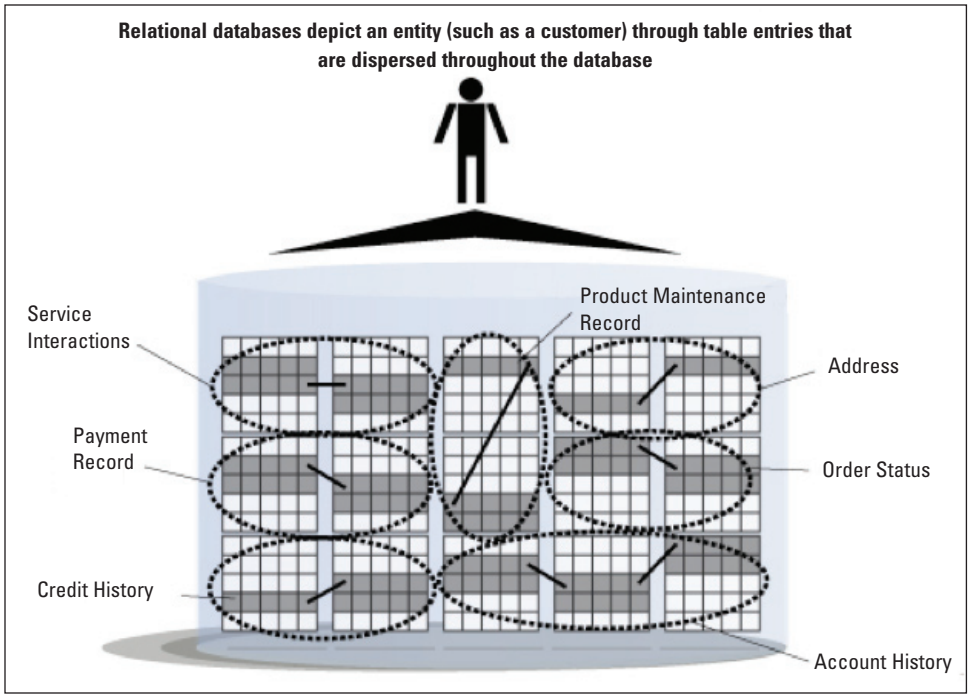

Figure 3: A business entity consists of many interrelated elements as rendered in a database

Such design goals mean that even the most basic business entities will not be found in a single logical or physical location in a relational database. A customer account, an employee record, a product bill-of-material, or a supplier relationship must be assembled from entries that are strewn across multiple tables. Such structure makes perfect sense for data integrity, but it proves a tremendous barrier to delivering real-time responsiveness.

Getting to Relational Data

Performance of real-time applications in such an environment often suffers "death by a thousand cuts." Middle tier data access is a resource- and time-consuming process that typically involves:

- Creating data structures for a relational client API.
- Formulating SQL client requests, issuing calls to the appropriate RDBMS client library.
- Having the relational client library optimize the request for client/server communication and send a query via inter-process communication (IPC) to a relational server.
- The relational server must take the query and optimize it, analyzing which relational indexes might be used to optimize the query.

- Locks are acquired, the query is executed, result sets formulated, and a results list created.
- The relational server then must execute another IPC to send the result to the requesting client process.
- The application code iterates through the result set, taking the required results and creating application objects.

The aggregate performance impact of that sequence of steps must be multiplied by the number of data calls within an application, the number of application instances, the number of users, the physical latency of the network, the input/output capacity of the database, and numerous other factors. Indeed, a cottage industry has sprung up within the database industry to provide the analysis, architecture, optimization, and tuning that addresses this object-to-relational mismatch. Yet the fundamental problem remains: the design of many applications requires data to be requested and transformed *as part of the application's runtime execution*, thereby imposing significant performance hits.

Bigger Databases = Bigger Challenges

Performance critical applications suffer the most when accessing data managed by large databases with sophisticated schemas. Enterprise product or customer databases can be extraordinarily elaborate. A major SAP deployment may involve databases that feature over 9,000 tables and 10,000 indices. PeopleSoft applications may incorporate as many as 8,000 tables, while Siebel CRM applications can use 1,300 tables and 7,300 indices.

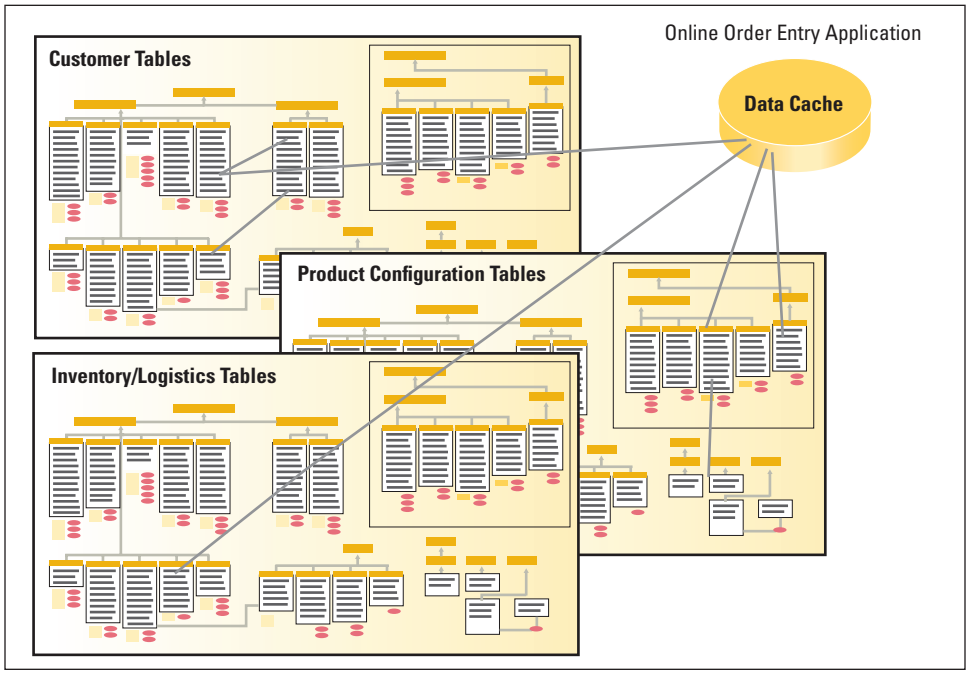

Figure 4: Corporate data often is organized according to needs of existing operational applications

Such complex, sophisticated database schemas are designed for the needs of core business applications and are not optimized for real-time access to small elements of data. And because these systems are so critical to line-of-business operations, corporate databases typically are tightly controlled and managed. Such carefully designed schemas cannot be changed without impacting the business operations.

To deliver the real-time responsiveness required of modern multi-tier applications requires a new set of data services. These data services must be delivered through a complementary architecture that leverages the data within relational repositories, but optimizes its delivery for real-time application requirements. Such a model understands the role of relational databases, but recognizes that no database system can do everything. Corporate databases serve the needs of traditional business applications, but need assistance to handle the transaction volumes, performance, and modern programming languages required by today's multi-tiered architectures.

Real Time Caching

Real-time caching solutions offer new data service capabilities that can meet these requirements. Among them are:

- **High performance** – Modern applications are transaction-intensive, demanding the responsiveness of in-memory data caching to support increased transaction volumes with minimal latency. Modern application servers often spread processing load across multiple machines in a way that is transparent to users. Data caching solutions must support the ability to coordinate caches distributed over multiple machines to ensure data integrity and reliability.
- **Efficient** – Real-time data caching must be efficient. Data caches should contain the data that is needed by the real-time application, but only what is needed. An efficient data cache culls the information needed by a new application without the burden of extraneous data that is part of the corporate database.
- **Distributable** – Much like modern technology architectures, modern businesses are highly distributed—because their operations are distributed or because functionality extends outside the organization to customers, partners, and suppliers. Caching architectures can mitigate the latencies that such deployments impose on performance. This argues for the availability of local caches that are situated close to the applications needing data.
- **Consistent** – There must be a consistent, accurate representation of the data that is mutually shared by the cache(s) and the corporate databases of record. A data cache that offers low latency, but inaccurate information is of little value. A cache that ensures accuracy at the expense of timeliness will fail as well. Effective data caching requires timely accurate representation of the data.

HISTORICAL PRECEDENT

This is not the first time that the needs of business have sparked demand for new classes of solution that can leverage the data managed in corporate repositories. Over thirty years ago, organizations sought to perform sophisticated analysis on the information within their databases. What emerged were solutions for data warehousing, analytics, and, more recently, business performance management. These technologies use enterprise data as the foundation for analytic and modeling functions that are beyond the capabilities of core databases.

The lesson is that a) no single database solution can serve multiple purposes effectively and b) new solutions must augment, not supplant, the core data repository. Similar arguments lie at the heart of the argument for data caching. General purpose databases are designed and tuned for broad enterprise requirements. They cannot accommodate those requirements and deliver the transaction rates and data integrity demanded by newer real-time imperatives.

- **Accessible** – Modern applications (with their object-orientation) and corporate databases (with their relational orientation) must work cooperatively to deliver the responsiveness required. Data caching solutions must reach a happy medium to deliver data to the applications in a format appropriate for its use.
- **Reliable** – Data caching solutions service the front line of the enterprise architecture. They must be available to support modern applications that operate 24 by 7.

ObjectStore has proven experience in responding to these requirements by offering enterprise class caching solutions built upon patented technology for customers like Starwoods, Orange UK, Delta Airlines, and Nomura International.

THE CACHE-FORWARD ARCHITECTURE OF OBJECTSTORE

ObjectStore is a patented real-time data caching solution designed to support applications where large numbers of transactions need access to data from enterprise data stores. With ObjectStore, application designers can create solutions that leverage the data managed within enterprise relational databases, yet avoid the data access bottlenecks that would result from servicing transactions by querying *directly* against the databases during transaction execution.

ObjectStore ideally operates in the middle tier of multi-tiered architectures. It caches enterprise data in a low-latency, memory-based architectural model that allows complete data consistency within and without the enterprise. Therefore, ObjectStore offers a unique combination of both data responsiveness and data accuracy that provides tremendous value to applications that are called upon to service ever greater numbers of users in ever more different ways.

Among the rich array of capabilities available with ObjectStore are:

- A patented Cache-Forward Architecture that delivers persistent data in a cache that spans multiple transactions, thus ensuring both the availability and accuracy of data.
- Distributed caching that scales solutions either through additional server instances or additional physical servers, while maintaining consistency across the caches.
- Supports language based persistent data models to eliminate the impact of coding via JDBC, ODBC, or proprietary APIs that will necessarily impose on runtime performance.
- Real-time cache responsiveness based upon in-memory access to data without the need to convert the data from relational to object format, or write and execute SQL code.
- Complete data integrity within the cache by ensuring enforcement of fundamental ACID properties for all transactions.

IN-MEMORY DATABASES?

The concept of an "in-memory" database for real-time data cache responsiveness appears attractive. But in-memory databases have fundamental limitations. Cache size is inherently bounded by the amount of memory available to the database. Since the cache resides in-memory, it also has no native persistence and, in event of a system failure, requires a complete and time-consuming rebuild of the cache unless redundant systems are used (thus driving up costs with two versions of both hardware and software operating in parallel.) Furthermore, providing tabular data, albeit in-memory, does not resolve the runtime requirement to convert the data into the application's object model. In-memory data still requires an object-relational transformation before it can be used by the application.

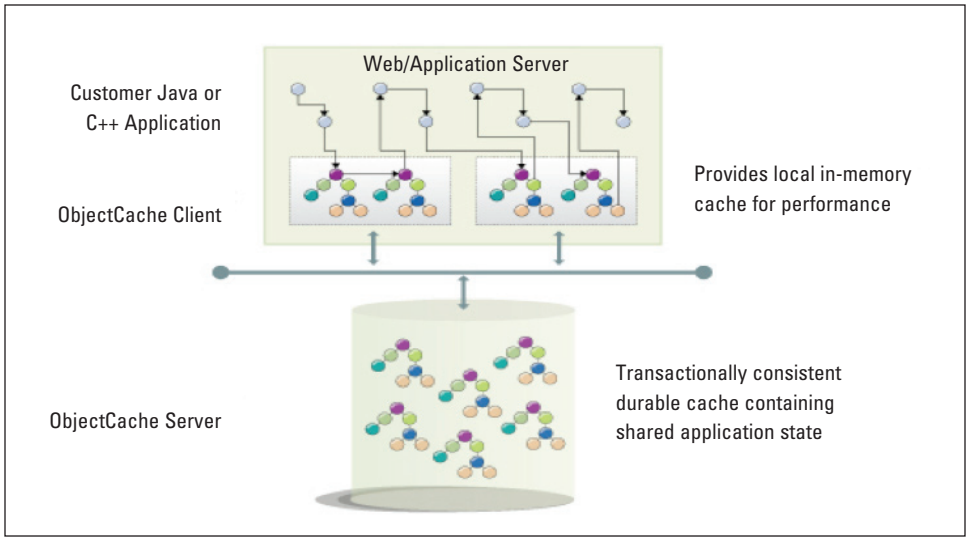

Figure 5: ObjectStore's architecture of persistent cache and local in-memory caches

ObjectStore performs these tasks automatically and transparently to the application, allowing an application developer to concentrate on the application's business logic and leave cache data management to ObjectStore. ObjectStore's design permits solution architects to incorporate relational databases into modern high performance applications, yet removes the constraints that object-relational mapping between disparate formats would normally have on the runtime execution of these solutions. Pre-populating a cache with the information needed to support these new applications eliminates data transformation from impacting application performance. With ObjectStore the data is already in the cache—and already in the object format needed by the application.

Delivering Data for Real-Time Transaction Support

In contrast to cumbersome runtime transformations of relational data, ObjectStore implements an innovative model that optimizes both the availability and utility of data in multi-tier architectures. ObjectStore removes SQL's intrusive processing overhead by allowing data to be optimized for the use case in question and moving it forward into the application's memory so that it is immediately accessible. This architecture mitigates the inherent performance impact of object/relational impedance, caching the data to achieve the optimum affinity for the application's runtime execution.

The Cache-Forward Architecture

Delivering data in a readily accessible format is but one of the virtues of the ObjectStore architecture. A key contributor to its low latency and data integrity is the patented Cache-Forward Architecture.[2] Cached data is placed close to the application execution

2. US Patents # 5,426,747; #5,649,139; #6,199,141; and #6,594,751

logic, allowing an application designer to reduce, if not totally eliminate, repetitive data requests that constrict system performance in multi-tiered applications. Rather than issue data requests to a server-resident cache across a local or remote network, ObjectStore maintains data in locally available caches where it can be accessed with in-memory speed.

Other caching mechanisms may provide in-memory data access, but delivering rapid data access—however fast the response—is NOT adequate *unless* the data that is delivered is assured to be accurate. Delivering inaccurate information quickly is hardly a virtue. What distinguishes ObjectStore is that, in tandem with its in-memory caching capability, it also guarantees complete data integrity by ensuring that each cache is consistent with across processes. Thus, ObjectStore's architecture ensures that timely, accurate, and up-to-date information is available to the application without requiring any coding contrivances by the application developer.

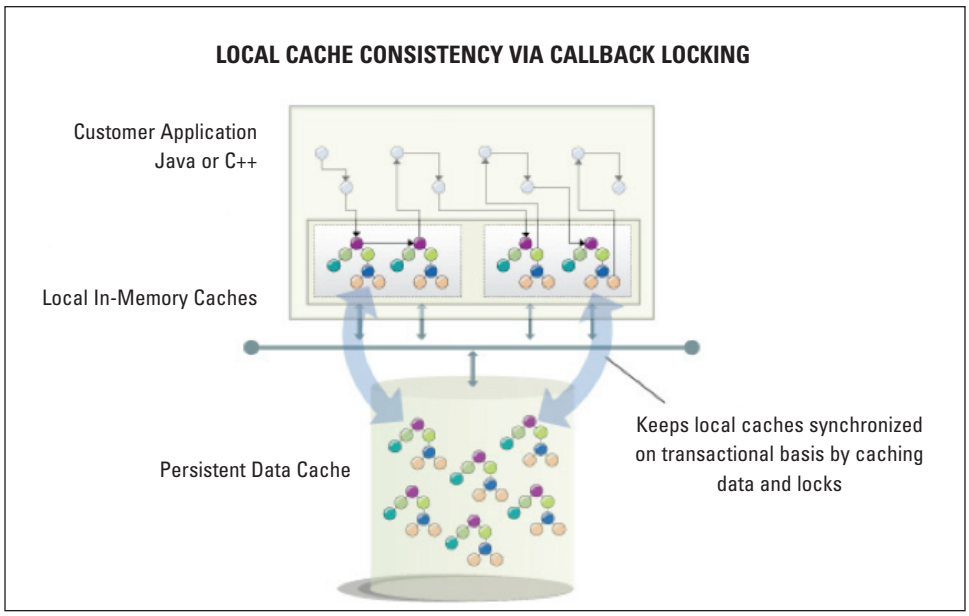

Figure 6: Local cache consistency via callback locking

To ensure cross-cache consistency ObjectStore supports a distributed cache locking mechanism, Callback Locking, that is fundamental to delivering a high performance, highly scalable caching infrastructure. By caching locks with the data in the application's memory, Callback Locking enables the cache to support a series of transactions against the same data without the need to retrieve data from the server cache or reacquire the rights necessary to execute subsequent transactions against the data. This capability is crucial to ObjectStore's ability to support large transaction volumes and sub-second interactive responsiveness to queries. Should a transaction executing against a local

cache change the data, all affected caches are refreshed with an updated copy fetched the next time the data is needed.

The Callback Locking mechanism is critical for real time systems, because few business processes are completed in a single transaction. There will be numerous data reads and/or writes that are nested within a single business transaction. The repetitive lock acquisition required of many applications by inefficient caching generates significant network traffic and database overhead that makes sub-second response impossible. ObjectStore's Callback Locking allows the memory cache to retain control of the data until the logical conclusion of the business event. In the event that changes to the data require updating of the local cache, that change is automatically propagated by the Cache-Forward Architecture.

Accelerated Performance with Virtual Memory Mapping

ObjectStore's ability to deliver rapid data access is further enhanced by a unique memory-based data mapping facility, the Virtual Memory Mapping Architecture (VMMA). ObjectStore manages persistent data in the same manner that an operating system manages virtual memory and paging. Data objects are mapped into an application's virtual address space, enabling persistent data within the cache to be accessed at in-memory speeds.

ObjectStore automatically detects references to data within the application and, if the data is not in memory, automatically transfers the pages from the local cache containing the referenced data to the application. Thus, ObjectStore automatically manages all persistent data without the need for explicit calls on the part of the application itself. This architecture frees the application developer from concerns about how to access the data and how to manage the data upon completion of the transaction.

Unlike caching solutions that present data in memory, but do not support transactions against the data (or risk data integrity to do so), ObjectStore delivers the requisite ACID (atomic, consistent, isolated, durable) properties that mission-critical applications require. ObjectStore also supports a high-speed, read-only cache access mechanism, Multiversion Concurrency Control (MVCC). MVCC is a special transaction mode that acts like a Repeatable Read isolation level (similar to a snapshot as defined by JDBC). The data accessed by the MVCC transaction can be read or written by other transactions without any lock contention due to the MVCC transaction. The user can control the lifetime of the MVCC transaction from a few hundred milliseconds to minutes.

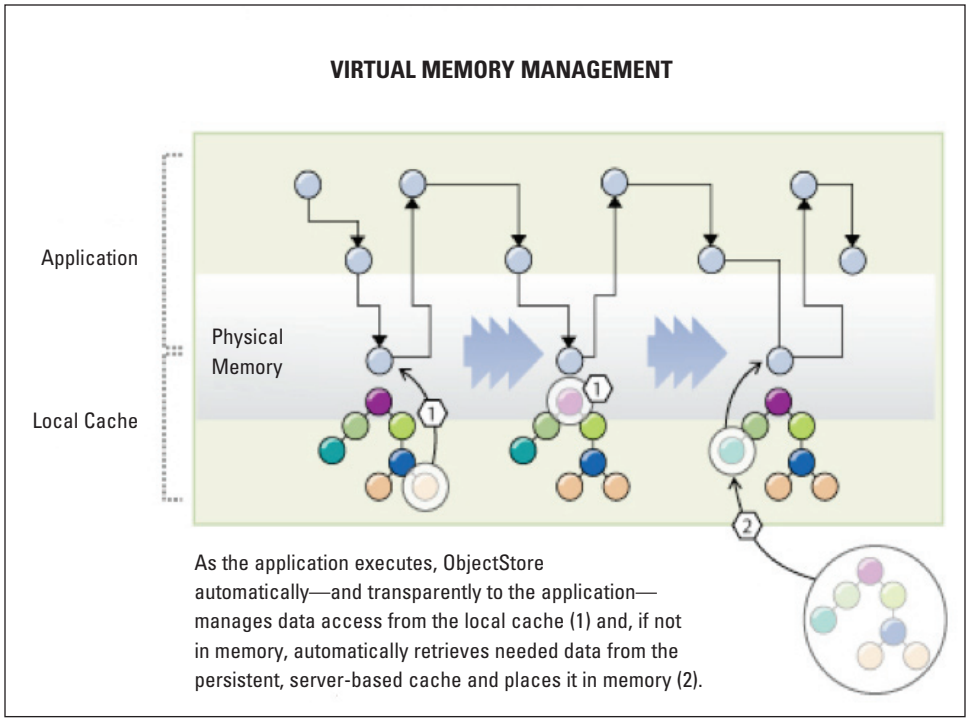

Figure 7: ObjectStore virtual memory management

MVCC greatly facilitates support of large transaction volumes in multi-user caching systems by permitting non-blocking read-access of cache data at the same time that writes are being performed. For high volume applications with significant data volumes that must be available to the cache (e.g. streaming data feeds from a stock ticker, flight positioning information, or a number of military applications) MVCC has proven a significant advantage to our customers. Both applications and end-users can access data in the cache simultaneous with ongoing updates that keep the cache refreshed.

Data Clustering

Cache performance is optimized by logically and physically grouping data within the cache. Designing high performance applications requires cache designs that are attuned to the physical location of data. Data that is logically associated, but physically dispersed, undermines cache responsiveness. Through clustering and partitioning ObjectStore allows an application designer to physically group data objects that are logically related. When a particular record is requested (e.g. a customer's bank record) all the data associated with that record can be moved to the in-memory cache without additional latency.

Thus, when the application's business logic calls for related data, it is likely to already be in the cache. When it calls for recently-used data (e.g. a screen refresh/change) for the same customer, persistent caching of the cluster enables this data to be accessible

as well. Data clustering, when coupled with in-memory caching, greatly minimizes the demands on the backend database by reducing the number of time-consuming calls for additional data during the execution of a business transaction. Eliminating unnecessary I/O requests allows ObjectStore to service high volumes of interactions.

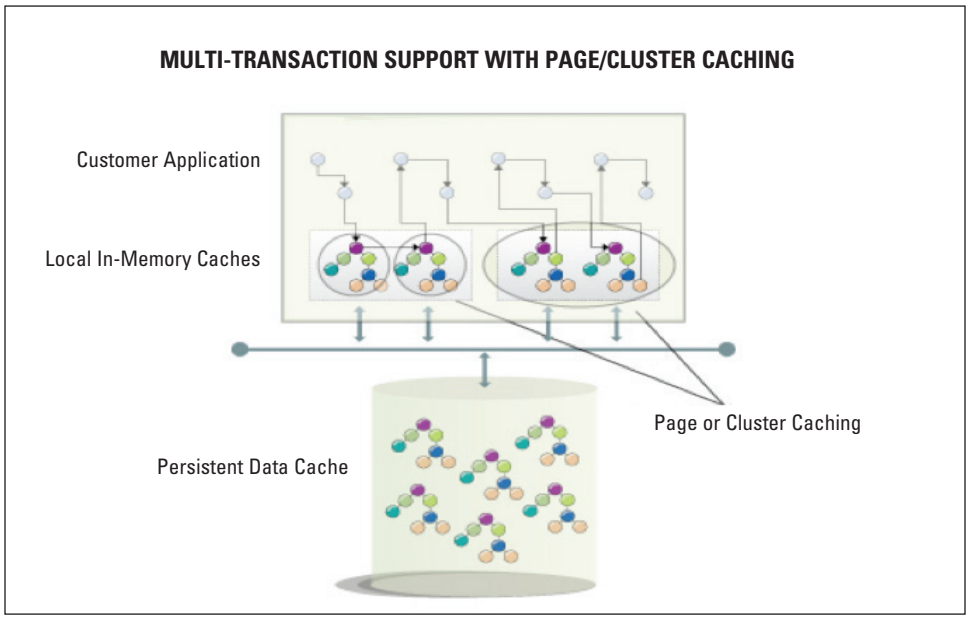

Figure 8: Clustering data within the cache

Integrating with Enterprise Data Stores

Corporate databases manage large amounts of data that are distributed amongst hundreds or thousands of different tables. Much of the data is extraneous to the needs of a middle-tier cache that is supporting a particular high volume application. Additionally, there is no assurance that the data needed by the cache is in close proximity to the relational database structure or location. To service a high speed application directly from such databases may require effort not central to the particular use case.

When ObjectStore is deployed with additional DataXtend products, the object/relational mapping may be done at design time by developers, database administrators or data modelers rather than in code at run time. This allows the application developer to implement application logic, not implement and maintain relational transformation code.

The resulting architecture:

- Automatically maintains consistent, up-to-date cache with the corporate database: changes in the relational database are propagated to the persistent cache in an efficient, low impact manner that offers unparalleled reliability and lifecycle maintenance.

- Offers a maintainable, agile integration with an unlimited number of heterogeneous data sources by integrating through a common exchange model. This isolates the effect of changes on the cache as new sources come online and business needs grow.
- Scales the middle tier more easily: Caches are automatically distributed and handle the management of state and transactions among the multiple tiers with no additional development effort.
- Partition and target the application of a cache for higher performance. Automatically move targeted data, specific to the services use case, from the result of joins across tables and data sources upon throughout the lifecycle of the cache. Deliver exactly the right data, live—as it is modified, from those sources to the cache. Don't just cache data for the runtime execution, cache the very process of mapping an in memory object model to the relational source.

In effect by deploying these products in tandem the benefits of a cache forward architecture are enhanced to seamlessly integrate with the enterprise information fabric. The DataXtend product suite creates a maintainable, low-impact, transaction-capable data cache that affords near-real-time access with unparalleled performance. It truly is a complete *object caching* architecture.[3]

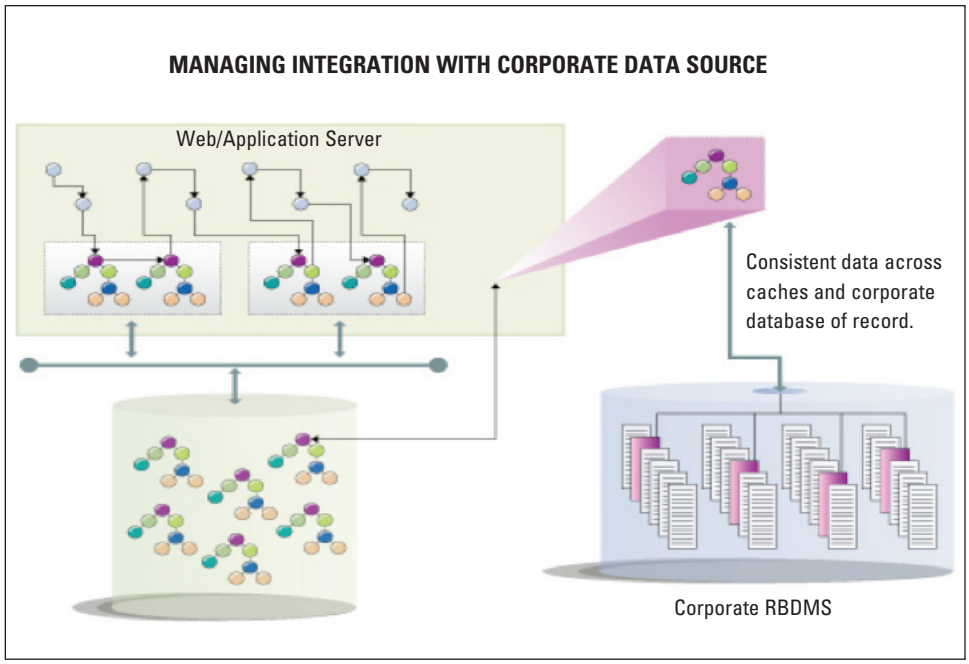

Figure 9: Integrating with the database of record

3. Additional information about these data integration products can be found at www.progress.com/dataxtend

OBJECTSTORE PERFORMANCE ADVANTAGES

Distributed Caching

ObjectStore's CFA makes it a natural for distributing caches in conjunction with a single back-end data store. Solution architects can distribute applications (and associated caches) to enhance system performance yet retain a single consolidated server-resident persistent cache to coordinate and ensure data integrity. Distributed caching delivers important scalability to real-time solutions by enabling organizations to add hardware, confident that their services, along with ObjectStore, can leverage the performance of those servers. Distributed caches also enable organizations to situate data physically close to the applications that need the data. Organizations that are geographically distributed (multi-site call centers, regional offices, outsourced operations, etc.) can mitigate the latency associated with wide area networks by placing data close to the application.

Each local instance of ObjectStore retains consistency with the other instances, as well as with the back-end data store(s). ObjectStore's design permits a transaction in one cache to check the status of a cached object across multiple distributed caches and, if necessary, lock the object to support write operations without need to contact the back end data store.

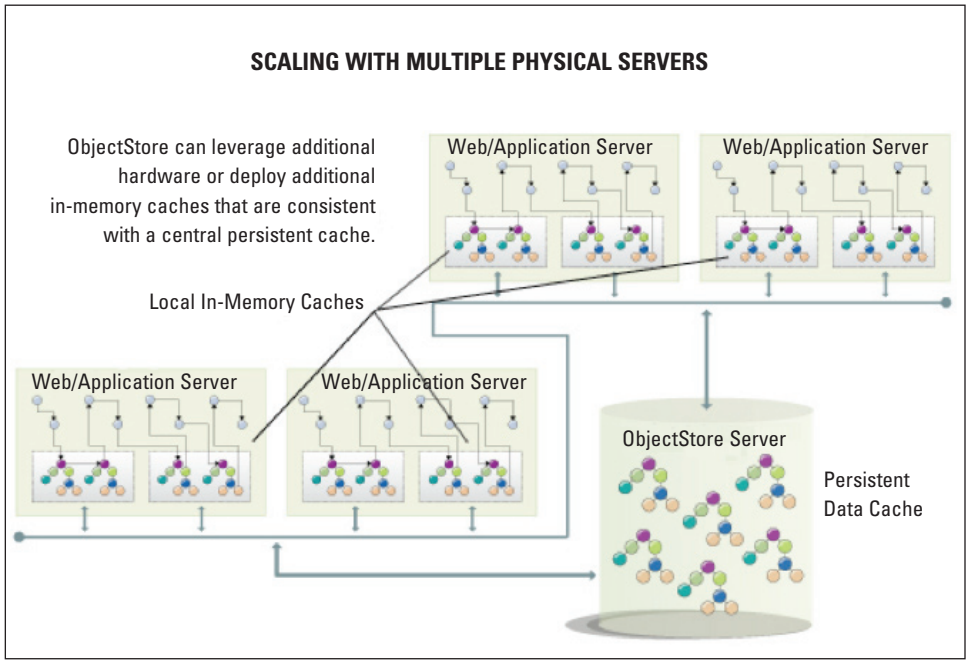

Figure 10: Multiple servers can leverage both hardware performance and physical proximity

Multi-Threading

The ObjectStore server is designed to use kernel threads, asynchronous I/O, and shared memory communication. ObjectStore clients use a Session Management Facility to support multithreaded applications. A single ObjectStore session can have multiple threads that share a database transaction. In addition, one process can have multiple sessions running multiple transactions. This Session Management Facility provides ideal support for multiple threads in large-scale application servers that are servicing numerous simultaneous requests. ObjectStore's session management delivers superior concurrency and performance. ObjectStore can also be configured as a single-process database with many threads, sessions, and transactions that operate concurrently within one application-server process.

Direct Support for the Application Object Model

ObjectStore supports data caching for applications developed in the language of your choice. The data schemas for the respective caches conform to the object model of the chosen language, with the class hierarchies stored in native formats. Standard libraries for managing collections of objects are supported directly in cache. The environment is completely intuitive to the application programmers and requires the developer to make no excursions into the orthogonal world of SQL, ODBC, or JDBC in order to manage persistent data within their applications. By leveraging an investment in corporate databases, and allowing the flexibility of development in the use-case appropriate native language, ObjectStore enables the best of both worlds in terms of tools. Application developers get to work with the most modern tools, while corporate database administrators retain control over one of the corporation's most valuable assets: data.

By encapsulating data within the native object constructs of the programming language and doing so on demand—ObjectStore eliminates performance-inhibiting factors like the runtime table joins that typify the use of relational databases in caching solutions. The object-relational mapping required to make relational data accessible to the application can all be handled in advance as part of the initial population of the cache.

At runtime, calls for data from within the application access data as objects, without the need to formulate tabular information into the appropriate object model. Neither reading of data from ObjectStore nor writing data requires any mapping code that would inhibit performance whatsoever. Such interface coding can represent anywhere between 25-60% of the development effort involved in creating high performance object oriented applications. By eliminating that code from the runtime execution of the application,

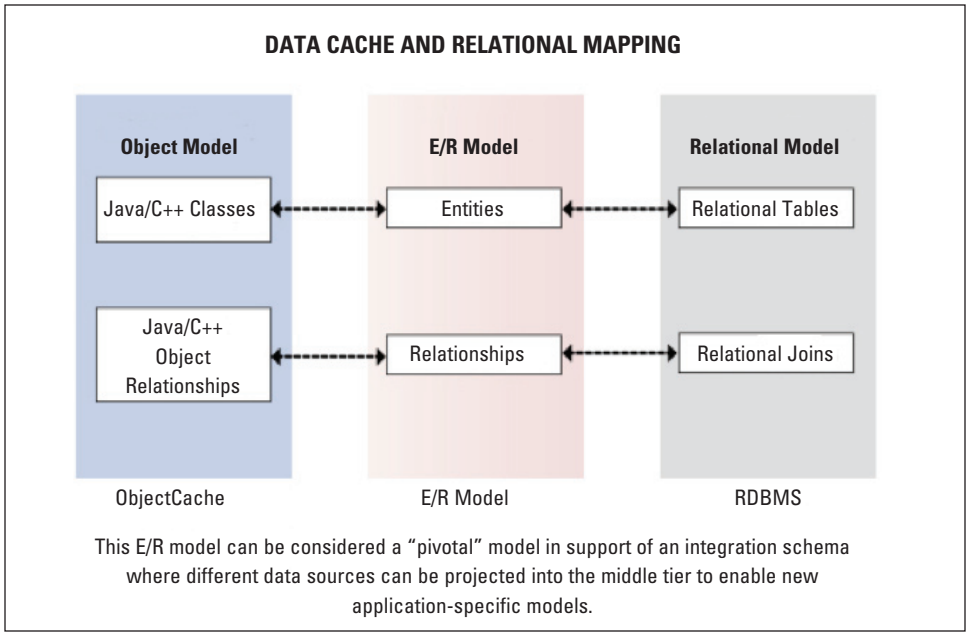

Figure 11: Object-relational mapping delivers data in a format that optimizes performance

ObjectStore greatly enhances performance. Eliminating such code also improves the overall development time. The reduction of coding effort greatly enhances application maintenance, given the portion of IT resources that are typically devoted to software maintenance operations.

Administration and Availability

ObjectStore's object model by definition requires neither database tables nor indices. Thus, ObjectStore eases many of the performance tuning chores that are common to relational data caching solutions and their need for optimized indexing for performance. For caching deployments that involve data inserts, deletes, or updates, the ability to operate without requirements for excessive index optimization, allows ObjectStore to function without daily administrative overhead.

This ease of administration also manifests itself in continuous operation. There is no need to take the system down in order to rebuild database indices, as is typically required of relational databases or relational caches. One customer using this technology in a distributed caching environment across a wide area network has commented that it was the closest thing to a fault-tolerant system that they have.

THREE CACHING EXAMPLES

The caching architecture of ObjectStore represents proven technology, deployed in major corporations throughout the world. Here are a few real world examples of customers that are utilizing this caching architecture to provide real-time responsiveness to their operational data needs.

Caching in Customer Service

A major communications service provider uses this solution to enhance the efficiency and effectiveness of its customer service operation. A subset of customer data has been culled from an Informix database and placed in an ObjectStore cache, powering a customer self-service application as well as an automated call-routing capability that assigns incoming customer inquiries to the appropriate call center.

Without caching from ObjectStore, access to a customer record in the corporate database would require a remote database query (with multiple table joins) across a wide area network. The composition of the query and network latency would have entailed lengthy query response times that could exceed ten seconds. Now, incoming calls are routed in sub-second time, without any noticeable latency. The Informix database remains the primary customer data repository and changes to customer data are propagated to the cache, ensuring that cached data is consistent with the primary database. The service provider maintains a single customer database—thus adhering to the customer care goal of a single customer view—yet is able to populate a series of data caches that power a distributed call center operation. This caching deployment extends the value of the corporate database by allowing its data to support inbound call management for many millions of monthly inquiries.

Caching in Online Sales

A Fortune 500 retailer has implemented an extranet to support its online marketplace. Operating as a middleman between buyers and sellers, the retailer needed a mechanism to ensure that inventory data was accurately represented to prospective buyers. Growing seller participation and increased transaction rates had created the likelihood that data presented to buyers was inaccurate, thereby potentially allowing buyers to purchase products that were no longer available.

To address the issue, the retailer implemented a data cache that front ends a series of Oracle inventory/fulfillment databases. Rather than execute queries against the Oracle database, prospective buyers view inventory availability information that is driven by cached data. Updates to the cache are spawned by triggers in the Oracle database. Without the data cache, queries of the main Oracle databases would have presented outdated information to prospective buyers and diminished satisfaction with the marketplace experience. With

caching technology from ObjectStore, participants in the marketplace process now are assured that the products shown as available are indeed available.

Caching in Financial Securities Trading

A large international securities firm uses cached data to support the real-time availability of customer and securities information as a critical part of the trade validation steps required to consummate transactions in the firm's settlement process. Customer and securities data is managed within a series of Sybase databases, but to enhance the real-time availability of the data, the securities firm caches Sybase information using ObjectStore technology.

Sybase remains the primary database, but data from the cache—sourced from Sybase—drives the trade validation process. This cached data also supports queries that allow traders to retrieve data about customer accounts or securities instruments. Use of data caching frees up Sybase, leaving it unencumbered by the back office processing required to resolve trade execution.

CONCLUSION

The heart of the value proposition for data caching and ObjectStore is to offer the data when, where and how you want it.

- Bring the data close to where it is needed and minimize transport latencies wherever and whenever possible. In distributed architectures that means providing data to the edge of the enterprise. ObjectStore delivers that through its Cache Forward Architecture.
- Ensure that data at the edge of the enterprise is a true, accurate, and consistent representation. ObjectStore delivers that through its Call Back locking mechanism and its enforcement of transaction integrity within the cache.
- Make the data available to applications in a format that is most accessible, executing the needed transformations outside the scope of the business event. ObjectStore meets that requirement with native support for your services to best meet the demands of transactionally intensive, stateful, multi-tiered architectures with complex object models.
- Leverage the performance advantages of hardware wherever possible to further reduce latency during application execution. Here, ObjectStore's support of local, distributed in-memory caches and its unique VMMA provide unparalleled performance advantages.
- Leverage existing data repositories and extend them to incorporate new usage models. Look to the suite of products offered by DataXtend to provide a complete solution that automatically captures updates from a data source of record and offers a maintainable, agile integration process that is based upon a common exchange model.

Distributed computing architectures require performance to be considered a critical component of design in order to support successful transaction-centric applications. Many sophisticated organizations have correctly appraised the situation and come to the realization that trying to deliver performance simply by scaling the hardware infrastructure is an ineffective and expensive solution. And cost is a material consideration. After all, in business solutions the laws of physics are matched only by the laws of economics.

More and more organizations are looking to the data caching capabilities of ObjectStore for the unique ability to offer the responsiveness of in-memory data access and the transactional integrity of a persistent data store. Together, those features deliver the responsiveness and scalability that is required by real-time enterprise applications.

PROGRESS SOFTWARE

Worldwide Headquarters
Progress Software Corporation, 14 Oak Park, Bedford, MA 01730 USA
Tel: +1 781 280-4000 Fax: +1 781 280-4095
www.progress.com

For regional international office locations and contact information, please refer to www.progress.com/worldwide