



Performance, Availability and Real-time Intelligence for Online Portals

- Achieving this triple convergence with the GemFire Enterprise Data Fabric (EDF)

Few would argue that we are in the midst of an online revolution. Just a quick peek at a stock ticker labeled GOOG speaks volumes. Most of this revolution thus far has been centered around consumer-centric (b2c) online businesses such as retail, travel and hospitality that have witnessed meteoric year-to-year growth as identified by the Goldman Sachs and Nielsen/Net ratings research reports. AMR Research estimates a 208B market potential for online businesses by 2007. But more recently, another trend has emerged. Enterprises in multiple industries including finance, retail, insurance have started to leverage the online channel for both intra-enterprise as well business-to-business (b2b) applications. These applications support core functions such as sales, customer service and information delivery. Such an online strategy enables even smaller firms to act big and global, while enabling the larger enterprises to become more nimble and deliver services with no bureaucracy. This increased online service delivery model can however become successful only if these portals are built upon a data infrastructure that can guarantee performance, availability and 'intelligent' data delivery. An Enterprise Data Fabric (EDF) is such an infrastructure that can bolster online environments.



TABLE OF CONTENTS

Going Online: Boon for the business, bane for IT?	3
Enterprise Data Fabric: A data architecture for online applications	5
High performance data storage	6
Heterogeneous data access	6
Data distribution and data consistency	6
Connectivity to backend data sources	7
Real-time Data Analysis and Continuous Querying	7
Three Paradigm Shifts	7
Deploying the GemFire EDF in web application architectures:	
5 Common Design Patterns	8
Pattern 1: Persistent data access layer between an application middleware platform and data sources	8
Pattern 2: Sharing data and state across clustered J2EE or .NET web application servers	10
Pattern 3: HTTP session state caching and replication	11
Pattern 4: Plug-and-play query result-set caching	11
Pattern 5: Continuous data analysis for event-driven web portals	12
Summary	14

GOING ONLINE: BOON FOR THE BUSINESS, BANE FOR IT?

Web applications provide an ideal channel for businesses to enhance their ability to serve their customers, employees, and partners better. For instance, online retail has been an area where this model has been embraced for several years now. Today, the range of services and products offered via the web is mind-boggling. In online trading, for instance, there are a slew of services that cater to both casual traders as well as power traders, who now have sophisticated online tools to wade through complex markets such as options and futures. On the business-to-business side, vendors utilize the web as a medium to service their clients through portal applications. This can to a great extent alleviate the pain that a typical business entity encounters in sharing real-time information to its internal and external users. In all these application scenarios, the success of the online service delivery model hinges on the ability of a web application to ensure the following three foundational tenets:

Personalization - A user is made to feel that the portal or the web page has been built keeping his or her specific behavior and needs in mind. The system operates in a perpetual learning mode.

Intimacy - It is often a comforting feeling for a user to know that there is a resource that provides up to date information on a 24x7x365 basis. This creates a stronger bond between the customer and the service provider.

Scale - A web applications helps a business entity cater to customers that it doesn't see, hear or know even exist. Thousands of customers in increasingly far-flung locations are reached and serviced consistently in a scalable manner.

While business executives reap the benefits of serving customers, partners and employees with such an online model, their IT counterparts have the rather unenviable task of building, deploying and managing these applications in a scalable and reliable manner. The fact that only 14% of newly developed web applications meet response time requirements (source: Gartner research) comes as little surprise, as the infrastructure part of the web application equation has not been adequately addressed until late in the deployment cycle.

IT groups face a litany of challenges when it comes to deploying large-scale online applications. First, these applications have to be designed and deployed to handle and scale to unpredictable loads, where the duration and nature of client access is unknown. For instance, the poor response times, or even worse, the unavailability of retail web sites operate during holiday seasons is often a consequence of infrastructure deficiencies. Second, the volume of data, especially dynamic, and in many cases fast-changing data, that needs to be managed within online environments such as portals can cripple most infrastructures. Third, end-users expect better or at least no worse levels of service with online applications as compared to the offline alternatives. What a large number of employees once handled must now be delivered automatically via the web without any degradation in the quality of service. Barriers to entry

are low in the online world, and as a consequence IT teams are challenged with deploying more and more innovative, value-added services to entice customers and prevent them from defecting to a competitor.

With these goals and constraints in mind, what infrastructure factors should an IT architect keep in mind when designing web systems that guarantee pleasant customer experience, handle sporadic traffic and support more transactions? The following represent 3 key factors that merit serious consideration:

- **Performance:** This involves delivering the necessary data (from backend sources) to a user. It also means updating backend data upon user changes and making these changes visible to other users with minimal latency. For instance, a travel reservation site that suffers from data access latency will more often than not be providing users with stale content ("The airline itinerary you selected is no longer available"! - Does this message look familiar). Performance becomes even more important when there are thousands of concurrent users pounding your web cluster. For instance, the look-to-book ratio, a common metric used in the online travel reservation industry to compare user visits to number of actual transactions, has increased from 50:1 to 300:1 just over a few years. Thus, high performance in itself isn't sufficient, it needs to be scalable even under peak loads. Most traditional data sources are not designed to handle the level of concurrent activity that an online world presents, and the usual remedy - throw more hardware at the problem. Though this approach provides temporary relief, in the long run it causes more harm than good and significantly increases cost of ownership.
- **High Availability:** Another major challenge with online applications is the ability to guarantee 100% uptime. Downtime often results in loss of business or loss of customers to competition. In most web architectures, user session objects are used to store a lot of customer state information. Hence, it becomes extremely important to preserve this session state in the event of one of the nodes in your cluster failing. Session replication to a cluster of nodes without performance degradation becomes critical. The other option is to adopt a sticky load-balancing mechanism in which case you have to be absolutely confident that none of your web application servers will ever go down.
- **Real-time Intelligence:** In the world of cut throat competition and low cost of witting vendors, providing intelligent and context sensitive services to online users becomes a key differentiation. Web portals that are able to operationalize customer information that is often safely tucked away in data warehouses, and correlate the same information to real-time user inputs or activities are "stickier" when it comes to customer retention. They enhance the feeling of personalization as users are presented information of likely interest to them based on their past history and current context. To achieve this new level of sophistication, intelligent data placement strategies and real-time data analysis techniques have to be blended at an infrastructure level. Such techniques provide a novel approach to support context-based services and product upselling in areas such as eCommerce and online gaming/gambling.

ENTERPRISE DATA FABRIC: A DATA ARCHITECTURE FOR ONLINE APPLICATIONS

60-70% of the latency and scalability problems of online infrastructures are outside of the application tier and in the data sources layer (source: Meta Group). Given this statistic, it becomes obvious that a robust data platform is absolutely essential to ensure a pleasant, consistent and reliable user experience with online applications. Industry pioneers are defining an **Enterprise Data Fabric (EDF)** as a solution to address this need.

Conceptually, an EDF is a pervasive and responsive information infrastructure that manages data predominantly in memory and across distributed nodes to enable the delivery of the right kind of data at extremely low latencies to handle transactions, analytics and decision-support in online applications. From a physical standpoint, an EDF (shown in Figure 1) can be envisioned as a data grid that provides high performance, in-memory data management for a online environment spanning the web tier, middleware-tier (application server) and data source tier. It harnesses memory and disk as necessary from physically distributed servers into a single, extensible enterprise-wide distributed cache. This enables any process to reliably share, persist, replicate, transform, analyze, and synchronize large volumes of data across the grid in real-time.

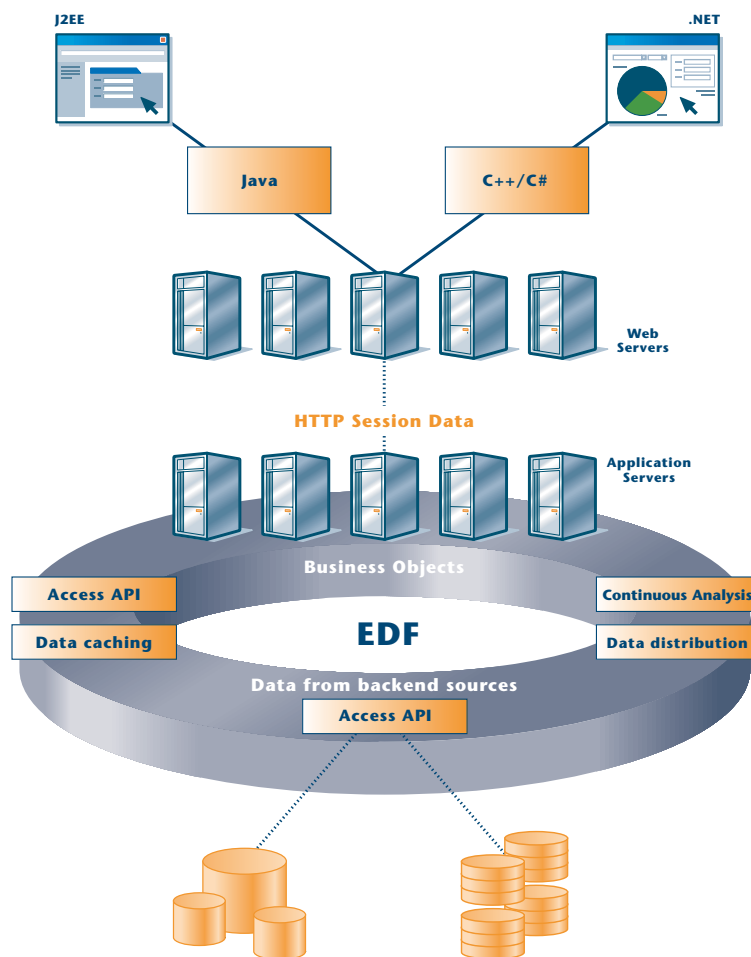


Figure 1: EDF in an online portal architecture

The GemFire product suite from GemStone Systems is an EDF that can bolster web application and portal deployments with its following key features

High performance data storage

GemFire offers multiple topologies¹ (such as peer-to-peer, client-server, hierarchical) to cache data in optimized in-memory (RAM) data structures, close to the point of use. This guarantees blazing data access performance and scalability by eliminating disk I/O. Furthermore, by pooling memory across distributed hardware nodes, the RAM limitations of a single node are overcome. Hence, large volumes of frequently accessed data (e.g., customer profile data, authentication/authorization data, catalog information, etc.) or transient entities like user sessions can be held in the EDF layer to ensure instantaneous data access and avoid expensive database roundtrips. GemFire also provides sophisticated in-memory data replication (n-way) that prevents data loss even when applications fail. Replication also serves as an in-memory load-balancing scheme across a cluster. Additionally, data that is held in GemFire can be overflowed to disk using algorithms such as LRU or MFU, or persisted to disk or a database for recovery upon failure. Such a strategy is far less expensive and more efficient compared to fail-over and recovery models currently used in online infrastructures.

Heterogeneous data access

In addition to offering native storage models for data types such as objects (Java, C++, C#), XML documents, JDBC result-sets and relational tables. GemFire also provides a variety of convenient APIs to access the data held in the fabric. Java applications can use a JCache (JSR-107) compliant API and the Object Query Language (OQL) to access and query the data. C# and C++ applications can use an API that is compliant with those environments. XML applications can use interfaces such as XML:DB, Xpath or SOAP to access XML documents that are stored in the fabric using an optimized Document Object Model (DOM) representation. In section 4 of this paper (Deploying the GemFire EDF in online portal architectures: Design Patterns), we will discuss the use of GemFire for transparent JDBC result-set caching for scaling data access performance. With this diverse API support GemFire can serve as a common, distributed data infrastructure across web application environments that use a combination of application platforms such as J2EE and .NET.

Data distribution and data consistency

In order to support portal deployments that span multiple clusters, GemFire provides a high-speed transport and membership management layer that utilizes multiple protocols such as TCP/IP and Multicast. This layer enables on-demand delivery of data across a distributed system in a reliable fashion, quite like JMS. Data distribution in GemFire may be push-based (changes propagated on update) or pull-based (changes propagated only upon a request). Depending on the specific environment, applications can use asynchronous data distribution when strict consistency is not required or when concurrent processes have no data conflicts. Applications

¹ Please refer to the GemFire Enterprise Technical White-paper available at <http://www.gemstone.com/download> for more detailed information on the cache topologies supported.

that need delivery guarantee can use synchronous distribution with acknowledgements. GemFire also supports changes to one or more data entities as a single atomic unit of work using a built-in distributed transaction service. Further, GemFire provides a distributed event notification service where updates to a data element are received through an event-listener framework.

Connectivity to backend data sources

GemFire provides a simple set of plug-in interfaces to enable connectivity with backend data sources such as databases, applications, etc. GemFire provides an "out of the box" plug-in for connecting to an enterprise JMS bus. GemFire also provides integration with O/R mapping tools such as Hibernate and Kodo from Solarmetric (BEA). Web application developers implement a simple interface called 'CacheLoader' to load data from an external source into a GemFire cache. A loader is automatically executed when an object lookup in the cache results in a miss. GemFire takes care of managing and distributing the object to other cache nodes in accordance with the configured policies. For synchronizing changes to objects in the cache with a data source, the plug-in provides two additional interfaces, 'CacheWriter' and a 'CacheListener'. A 'CacheWriter' enables "write-through" caching and is used to synchronously write changes to the data source before applying the change in the distributed cache. A 'CacheListener' on the other hand, enables "write-behind" caching where the change is first applied to the cache and then asynchronously applied to the data source. Loaders, writers and listeners can also be executed remotely. For instance, a cache miss on a particular node can invoke a cache loader on another node, which is connected to a backend data source.

Real-time Data Analysis and Continuous Querying²

In addition to storing and virtualizing static data elements, GemFire EDF also provides the ability to manage fast-changing data and real-time event-streams such as stock price data, retail point of sale data, or customer web clicks. GemFire supports a continuous querying model, where new events are constantly correlated with other static data sources and analyzed against pre-defined queries or patterns of interest. This model is optimized to rapidly determine queries affected by a particular event and notify relevant web applications via callback interface with almost no latency. With GemFire, predefined patterns of interest can be analyzed, or new scenarios or queries dynamically created in real-time. This feature is extremely useful in providing context-sensitive real-time information to online users to enhance their web experience.

Three Paradigm Shifts

The preceding commentary on the key features of GemFire EDF provide an ideal platform to discuss three data architecture paradigms that make an EDF essential in environments such as online portals, large-scale grids and SOA.

² Please refer to the GemFire Real-time Events Technical White-paper available at <http://www.gemstone.com/download> for more detailed information on GemFire continuous querying and analysis capabilities.

- The first paradigm reflects the move to a distributed storage and persistence model as opposed to a centralized model that is typical of database management systems. With applications, hardware and processes becoming more and more distributed, storage and persistence have to be handled close to the data consumers for optimal performance and scalability.
- The second paradigm relates to an active data management model, wherein applications are provided access to relevant subsets of data (active subsets) and the persistence model manages that data only for the duration of a particular business transaction. After the completion of the transaction, the data is held in a backend system of record such as a database. Data is also stored in a format that is directly accessible by an application without any transformation overhead.
- The third paradigm deals with the notion of treating data as a dynamic entity and managing 'data in motion.' In addition to the large volumes of data stored in standard repositories, enterprises these days need to deal with real-time dynamic data streams such as credit card transactions. It becomes a critical infrastructure requirement to manage such fast-changing entities and also distribute and propagate updates across distributed applications.

DEPLOYING THE GEMFIRE EDF IN WEB APPLICATION ARCHITECTURES: FIVE COMMON DESIGN PATTERNS

The breadth of features offered by GemFire EDF makes it a good fit for several different architectural use-cases within a portal or a web application environment. The following describe typical patterns of using GemFire in such environments.

Pattern 1: Persistent data access layer between an application middleware platform and data sources

One of the typical areas of use for GemFire is as a large volume, distributed data cache that acts as a high-performance façade to backend data used by online applications (stand-alone or deployed within an application platform like J2EE or .NET). Current approaches that rely on using the database as an operational persistence layer fail from a performance and scalability standpoint leading to end-user frustration and possible defection to a competitor. Data management capabilities offered by more recent versions of application servers lack the levels of sophistication needed to support large volumes of data across a distributed cluster leading to chronic infrastructure issues.

To address these issues, the GemFire client-server cache topology shown in Figure 2 has often proven effective. Data from backend sources is pre-loaded using cache loaders into a distributed set of cache servers. Cache servers may also invoke a cache loader on demand upon a cache miss. Large data volumes may even be partitioned into multiple physical machines, but presented to a client as a single logical entity. Java, C++ or C# applications can connect to a cache server and also maintain a local client cache, which holds subset of the data being held in the cache server. Application may also choose to be a cacheless client and access all the data from the server cache via simple get/put APIs.

Client-server connections are handled via TCP/IP, and are load-balanced across multiple cache servers for scalability and automatic fail-over (hot standby). To make efficient use of CPU and bandwidth across thousands of clients, client caches can specify 'interest lists' or specific data entities that matter to them, so that only cache change notifications related to those entities are propagated to them. In general, cache updates are communicated bi-directionally from client-to-server and vice-versa. Moreover, data updates are propagated to the backend sources or other system members via listeners (asynchronous) or writers (synchronous). Both client and server caches can persist or overflow data to disk or replicate data to backup nodes to avoid data loss on node failure. Such a combination of in-memory data replication strategy coupled with disk persistence is an extremely cost-effective and more efficient alternative to expensive database replication and high availability solutions typically deployed in large-scale portal environments. Data overflow completely eliminates out-of-memory errors from applications servers even when they cache large amounts of data. This design pattern is ideal for managing large data volumes in a scalable fashion across a cluster of application servers, without burdening these servers with a large caching overhead, while at the same time removing the load on backend systems.

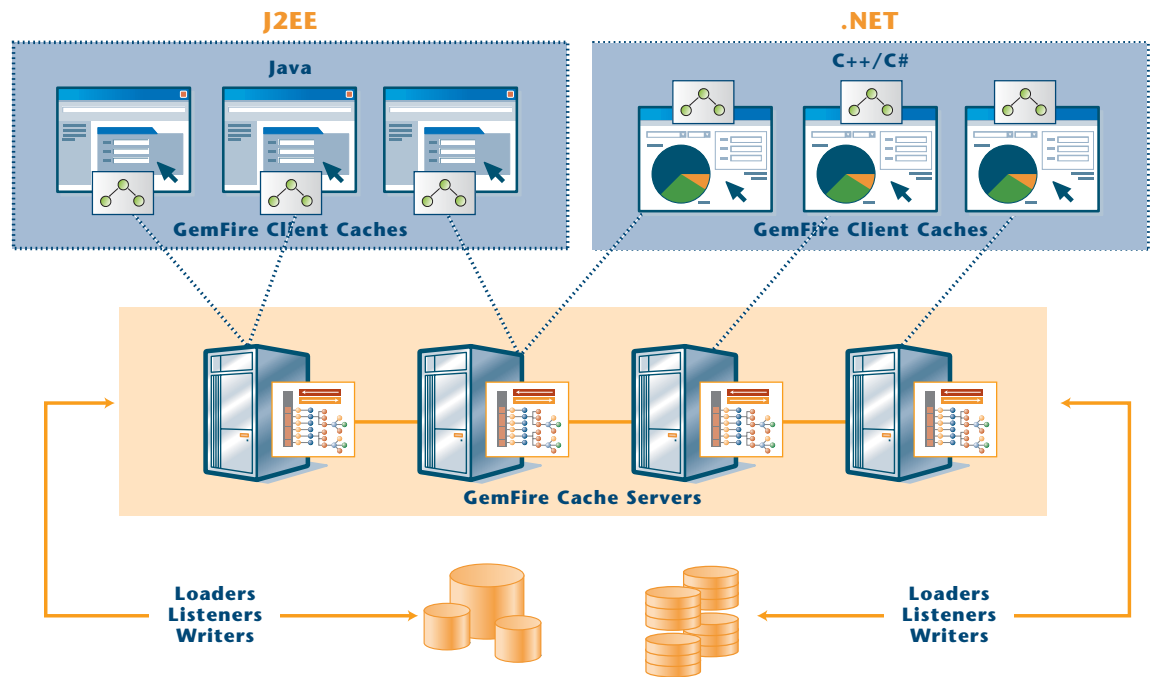


Figure 2: GemFire EDF client-server caching topology as a persistent data access layer

Pattern 2: Sharing data and state across clustered J2EE or .NET web application servers

As more and more application servers continue to manage state within their JVMs or C# processes, data management across a cluster needs to be sophisticated. State management functions offered by application servers are CPU-intensive and are unreliable unless used in conjunction with a protocol like JMS. However, JMS adds significant latency, and computational overhead to the data sharing process. To address this problem, GemFire offers a robust peer-to-peer cache topology (shown in Figure 3) that can be deployed not just across a single homogenous cluster, but across multiple heterogeneous clusters serving different classes of functions such as content management, billing, data transfers and order management. These clusters may even be spread across a Wide Area Network (WAN). With this peer-to-peer model, Java, C++ or C# applications can embed caches within their application process memory and enjoy blazing fast data access. These caches can be configured as local or private caches, or as replicated caches (for fail-over), or as partitioned caches, which for instance can hold a large volume of data (even hundreds of gigabytes) across multiple JVMs in a J2EE cluster and guarantee data delivery with at most one network hop. For instance, user security information and user authorization credentials can now be maintained in memory across a cluster and delivered to a particular application process on demand. All these cache types can be queried using OQL and also participate in distributed transactions.

Except for private or local caches, all other types are 'cluster-aware' and propagate any data changes to relevant peers. Data distribution and synchronization is handled via a reliable transport layer (TCP/IP or reliable UDP multicast), which provides messaging-like semantics for guaranteed and durable data delivery. The peer-to-peer transport layer is also sophisticated enough to handle network segmentations and gracefully handle those scenarios in a policy-driven fashion. It also has mechanisms to 'quarantine' slow application processes by queuing messages so that these processes do not degrade cluster performance

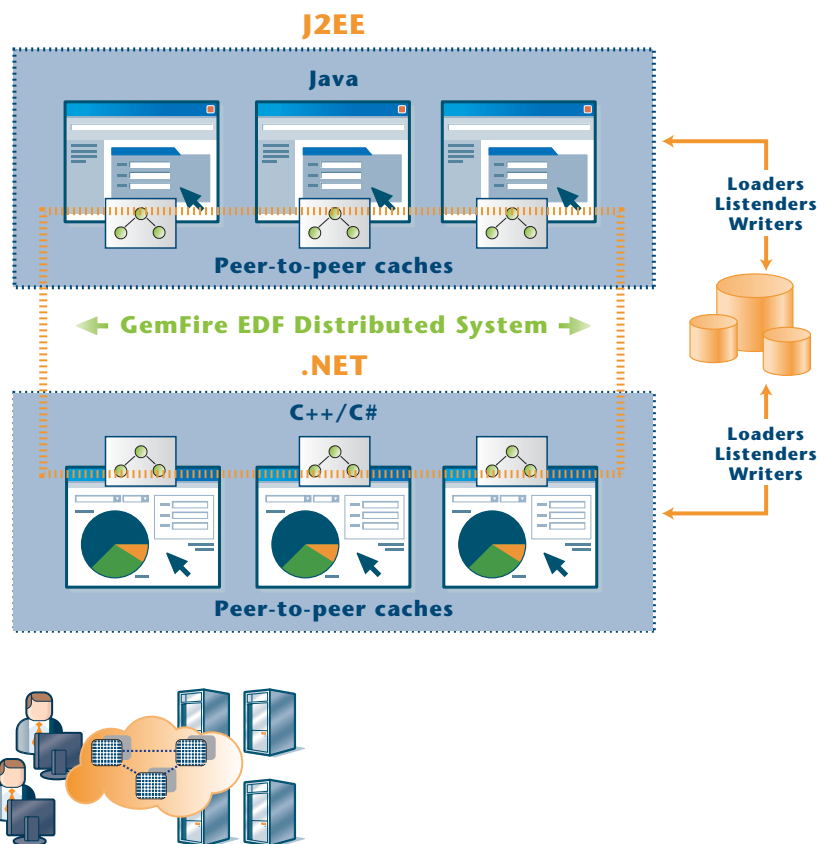


Figure 3: GemFire peer-to-peer cache model for clustered J2EE and .NET environments

Pattern 3: HTTP session state caching and replication

In an attempt to provide personalized services to online users, most IT architecture teams rely on storing state in the HTTP session objects. Over time, this approach results in session objects that become too large in size causing performance bottlenecks especially under heavy concurrent load with thousands of users. This problem can effectively be addressed using GemFire EDF. The different cache topologies discussed in the previous two examples apply to session state management across a cluster as well. By utilizing a cache server that holds user sessions across a cluster of application servers, the repeated serialization overhead is avoided, as all application servers access this data from the cache server, which itself can be replicated for fail-over. This approach is more efficient when compared to holding session data in a database, which causes unnecessary bottlenecks and increased wait-times especially under peak loads. Even when a peer-to-peer topology is used, GemFire offers an optional serialization protocol that is more efficient than the native Java serialization techniques adopted by typical J2EE servers. Session management capabilities offered by an application server involve replicating large session objects to each node in the cluster for every minor update to the session object. By utilizing the GemFire cache server model these expensive operations can be avoided.

By virtualizing access to session objects across an entire cluster, GemFire EDF enables a load-balancer to uniformly redistribute web server requests and not be constrained by sticky load-balancing. Further, user session objects held in a cache server can be accessed even by applications that are outside the web container and also be monitored via GemFire's console or through JMX MBeans. Another important feature in the context of session state management is GemFire's ability to overflow data to disk. In most online environments, there are several user sessions that are passive at any given point in time (i.e., a user is logged in, but not using the portal). With GemFire, these passive sessions can be automatically overflowed to disk using algorithms like Least Recently Used (LRU) or Most Frequently Used (MFU). When user activity resumes, corresponding session objects are automatically provisioned back to RAM by swapping it with another idle session object, if heap space becomes a constraint. In this fashion, cache server memory and application memory is prudently used only for active sessions. All these strategies greatly reduce the chances of performance degradation usually attributed to session management.

Pattern 4: Plug-and-play query result-set caching

Most J2EE-based web application environments access backend data from databases via the JDBC API. In most cases, there are significant latency issues associated with such data access especially while executing complex SQL statements under concurrent loads. GemFire EDF provides a pluggable mechanism to transparently intercept JDBC queries and cache their result-sets in-memory (Figure 4), so that performance is significantly enhanced and the load on databases is also greatly reduced. GemFire provides a specialized JDBC 2.0 driver that intercepts

query requests for caching and then redirects the call to the true JDBC driver if a database fetch is actually required. Application architects can review query statistics via an intuitive console that provides information on query execution times and frequency. Based on this visual tool, architects can decide to cache any query. Queries that are cached in any JVM of an application cluster can be accessed by any other JVM transparently. **No application code changes are necessary to accomplish this.** Queries can also be replicated or persisted to disk for high availability. Updates made to the database tables via the JDBC driver automatically lead to invalidation of data in the relevant caches. In this manner, database I/O constraints are removed from the path of web application transactions, and their scalability and performance greatly improved.

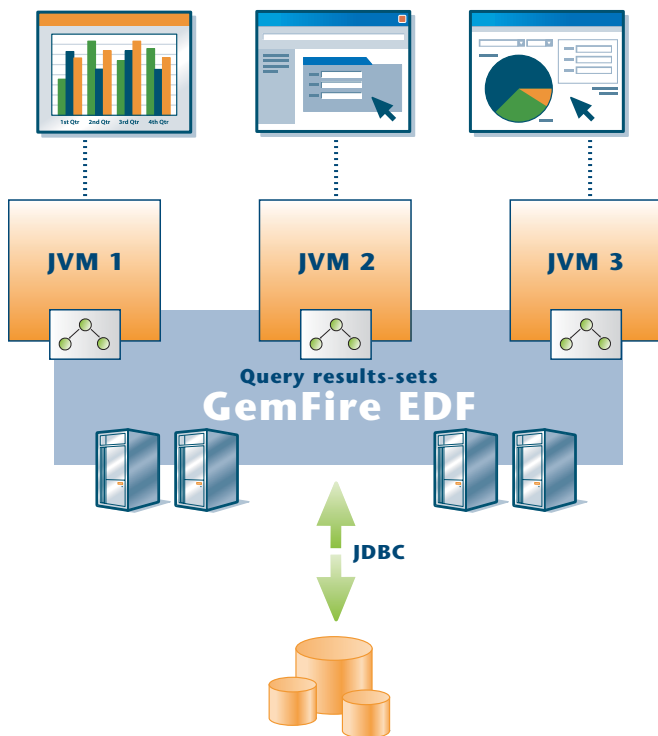


Figure 4: Transparent query result-set caching with GemFire

Pattern 5: Continuous data analysis for event-driven web portals

Online portals in today's economy have to account for real-time changes that happen in their ecosystem and also provide context sensitive information during the course of a user-transaction. Unless this level of intelligence is built into the infrastructure, users may be presented with stale/irrelevant information or may not be alerted to latest events happening within the enterprise relevant to their transaction leading to dissatisfaction. To tackle this issue, the GemFire EDF offers a solution GemFire Real-time Events (RTE) -

<http://www.gemstone.com/products/gemfire/rte.php> - which offers an event-driven data model that can be deployed to support web portal applications. As shown in Figure 5, Java C++ and

C# applications can register pre-defined queries of interest (described using SQL) with GemFire through a JDBC/ODBC interface. For instance, these queries may relate to customer margin violations in an online trading portal, or about new product offers based on particular criteria in an eCommerce site. Applications can also add queries dynamically at run-time. Real-time data streams such as stock quotes or product offers are constantly funneled into GemFire. Static data entities like product specifications or customer profiles are also stored within the fabric. Multiple GemFire RTE instances can be launched for load-balancing and fail-over purposes.

GemFire RTE support a continuous querying (CQ) paradigm wherein the queries registered by the applications are constantly evaluated against new incoming events and correlated with other data sources based on the query predicates. Based on this analysis, impacted queries and the owner applications are identified and callback notifications sent. These notifications include updates, which are essentially real-time changes to the result-sets associated with the registered queries. These updates can now be handled appropriately in the server-side applications as well as on the client-side applications (web portal). In this fashion, context-sensitive dynamic content can be injected into portals based on real-time changes that are happening outside of the web application container.

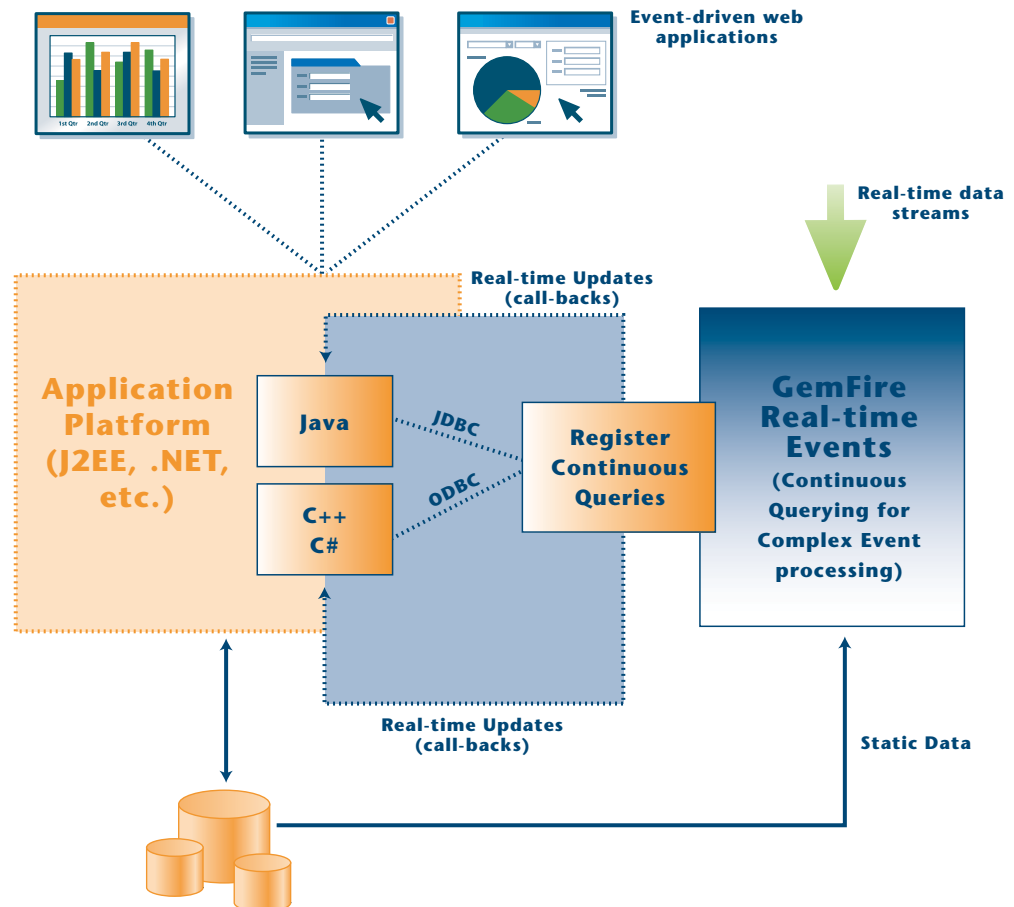


Figure 5: Event-driven web applications using GemFire

SUMMARY

Online business models are information-centric in nature, wherein data architectures are absolutely critical. As web-based channels become an increasingly significant source of customer and employee interaction, an EDF-based data strategy becomes essential to guarantee pleasant online customer experience, customer retention, drive more transactions, and increase revenues.

An EDF strategy for online infrastructures is anchored on the following salient principles:

- **Address performance and scalability challenges at an architecture level**

From an application development and maintenance standpoint, such an approach is orders of magnitude cheaper than constantly patching a fundamentally flawed infrastructure, which is typically the approach that most IT organizations are forced to follow.

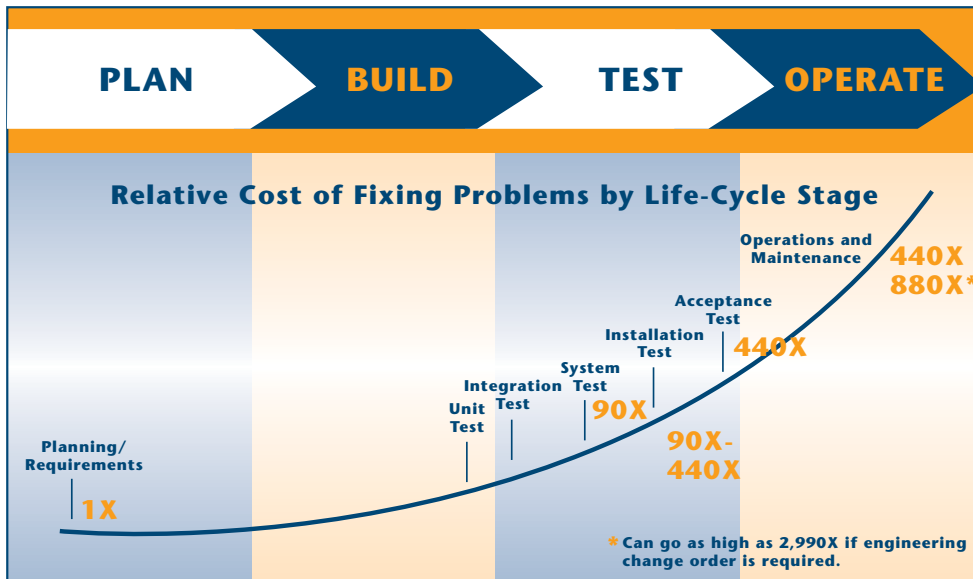


Figure 6: Cost of fixing problems along the IT lifecycle (source: Gartner)

Manage data close to point of use

Distributed, in-memory data management enhances performance and enables the delivery of relevant information to an online user in real-time.

Focus on making your portals faster and intelligent

The delivery of intelligent, personalized information to a user in a scalable manner is an important feature that is delivered through the distributed topology of an EDF, coupled with the ability to continuously query and analyze fast changing data.



Corporate Headquarters:

1260 NW Waterhouse Ave., Suite 200 Beaverton, OR 97006 | Phone: 503.533.3000 | Fax: 503.629.8556 | info@gemstone.com | www.gemstone.com

Regional Sales Offices:

New York | 90 Park Avenue 17th Floor New York, NY 10016 | Phone: 212.786.7328
Washington D.C. | 3 Bethesda Metro Center Suite 778 Bethesda, MD 20814 | Phone: 301.664.8494
Santa Clara | 2880 Lakeside Drive Suite 331 Santa Clara, CA 95054 | Phone: 408.496.0242

Copyright© 2005 by GemStone Systems, Inc. All rights reserved. GemStone®, GemFire™, and the GemStone logo are trademarks or registered trademarks of GemStone Systems, Inc. Information in this document is subject to change without notice.