TheServerSide.COM JAVA in ACTION
An Enterprise Java Conference and Training Experience.

Distributed Caching: Essential Lessons

Considerations for maximizing scalable performance and reliability in J2EE clusters

TANGOSOL

© Copyright Tangosol, Inc. 2005
All Rights Reserved

TechTarget
The Most Targeted IT Media

About: Overview

- **Application development considerations for achieving maximum scalable performance and reliability in clustered J2EE environments, improving scalability and scalable performance of applications through the use of clustered caching to reliably share live data among clustered JVMs in the application tier, providing transparent fail-over as a key element of uninterrupted operation, and reduced load on the database tier as a key element of scalability.**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

About: Cameron Purdy and Tangosol

- **Cameron Purdy is President of Tangosol, and is a contributor to Java and XML specifications**
- **Tangosol is the JCache (JSR107) specification lead and a member of the Work Manager (JSR137) expert group.**
- **Tangosol Coherence is the leading clustered caching and data grid product for Java and J2EE environments. Coherence enables in-memory data management for clustered J2EE applications and application servers, and makes sharing, managing and caching data in a cluster as simple as on a single server.**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Caching Topologies

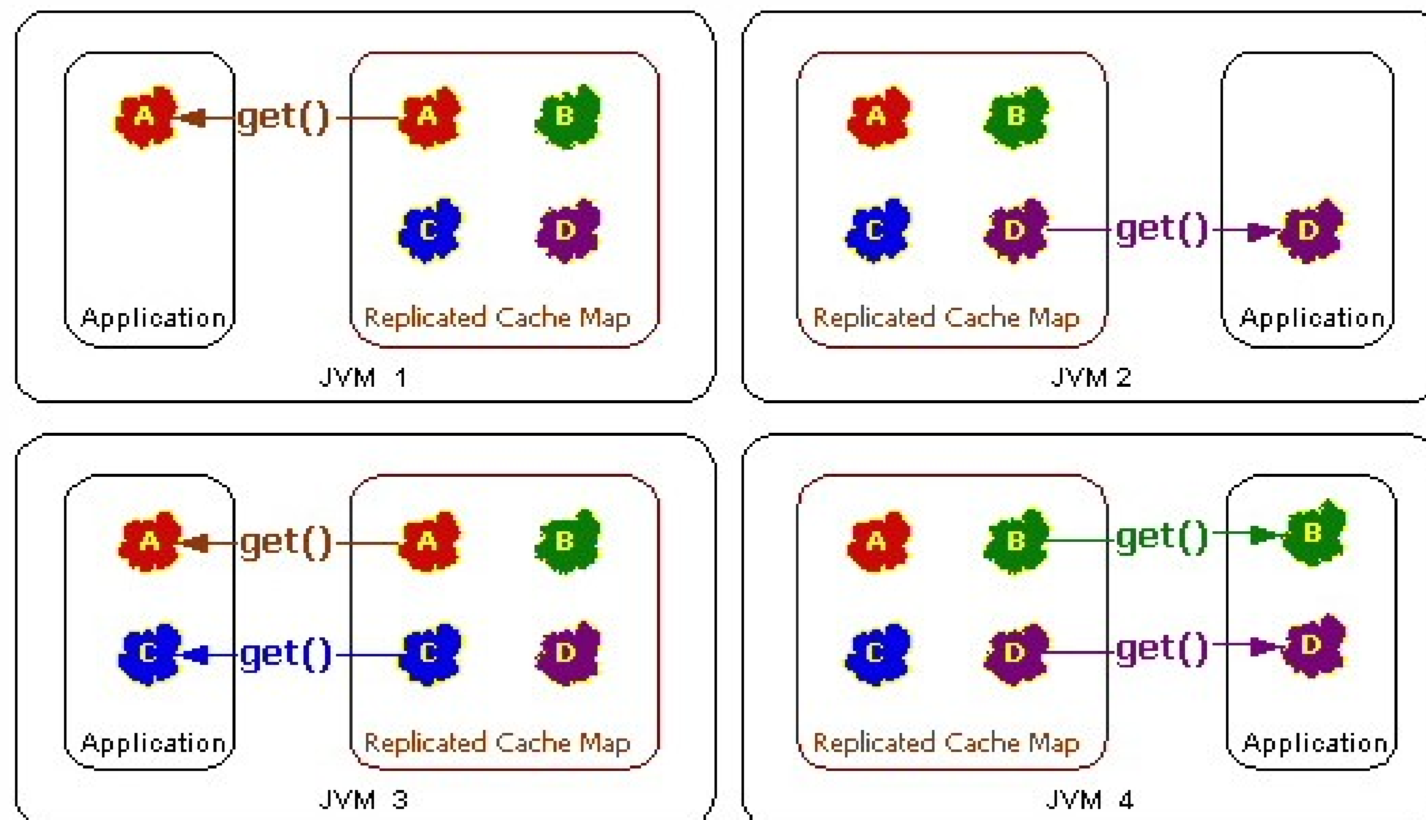
© Copyright Tangosol, Inc. 2005
All Rights Reserved

Replicated Topology: Description

- ***Requirement*** **: Extreme Performance. Zero-Latency Access.**
- ***Solution*** **: Fully Replicate data to all members of the cluster.**
- ***Result*** **: Zero Latency Access. Since the data is** ***pushed*** **(replicated) to each cluster member, it is available in each JVM for use without waiting. This provides the highest possible speed for read accesses.**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Replicated Topology: Read Access



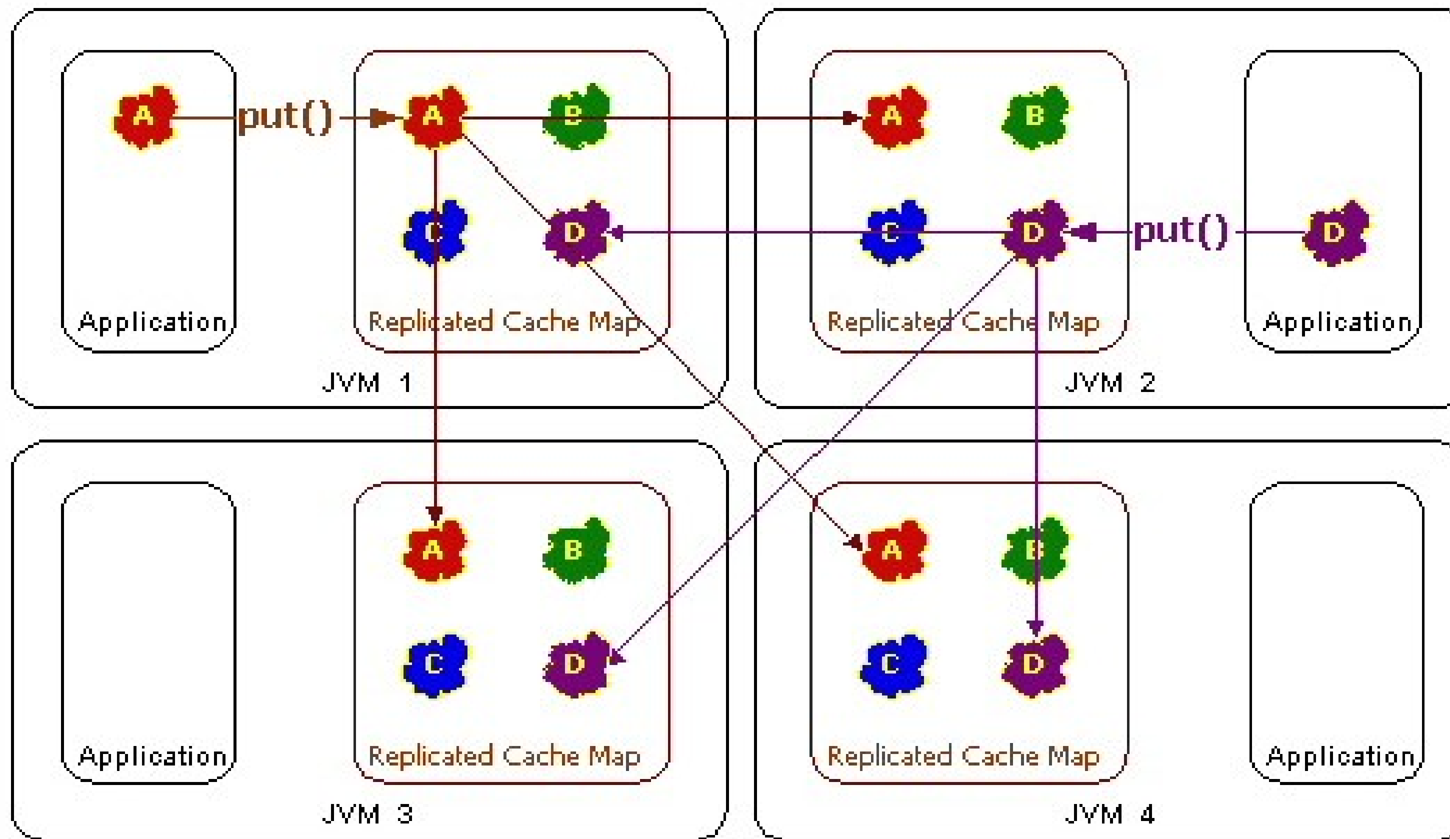
© Copyright Tangosol, Inc. 2005
All Rights Reserved

Replicated Topology: Limitations

- ***Cost Per Update*** **: Updating a replicated cache requires pushing the new version of the data to all other cluster members, which will limit scalability if there are a high frequency of updates per member.**
- ***Cost Per Entry*** **: The data is replicated to every cluster member, so Java heap space is used on each member, which will impact performance for large caches.**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Replicated Topology: Write Operations



© Copyright Tangosol, Inc. 2005
All Rights Reserved

Replicated Topology: Conclusions

- ***Performance*** **: Very good read performance**
- ***Scalability*** **: The scalability of replication is inversely proportional to the number of members, the frequency of updates per member, and the size of the updates**
- ***Uses*** **: Small read-intensive caches that benefit from a data "push" model**

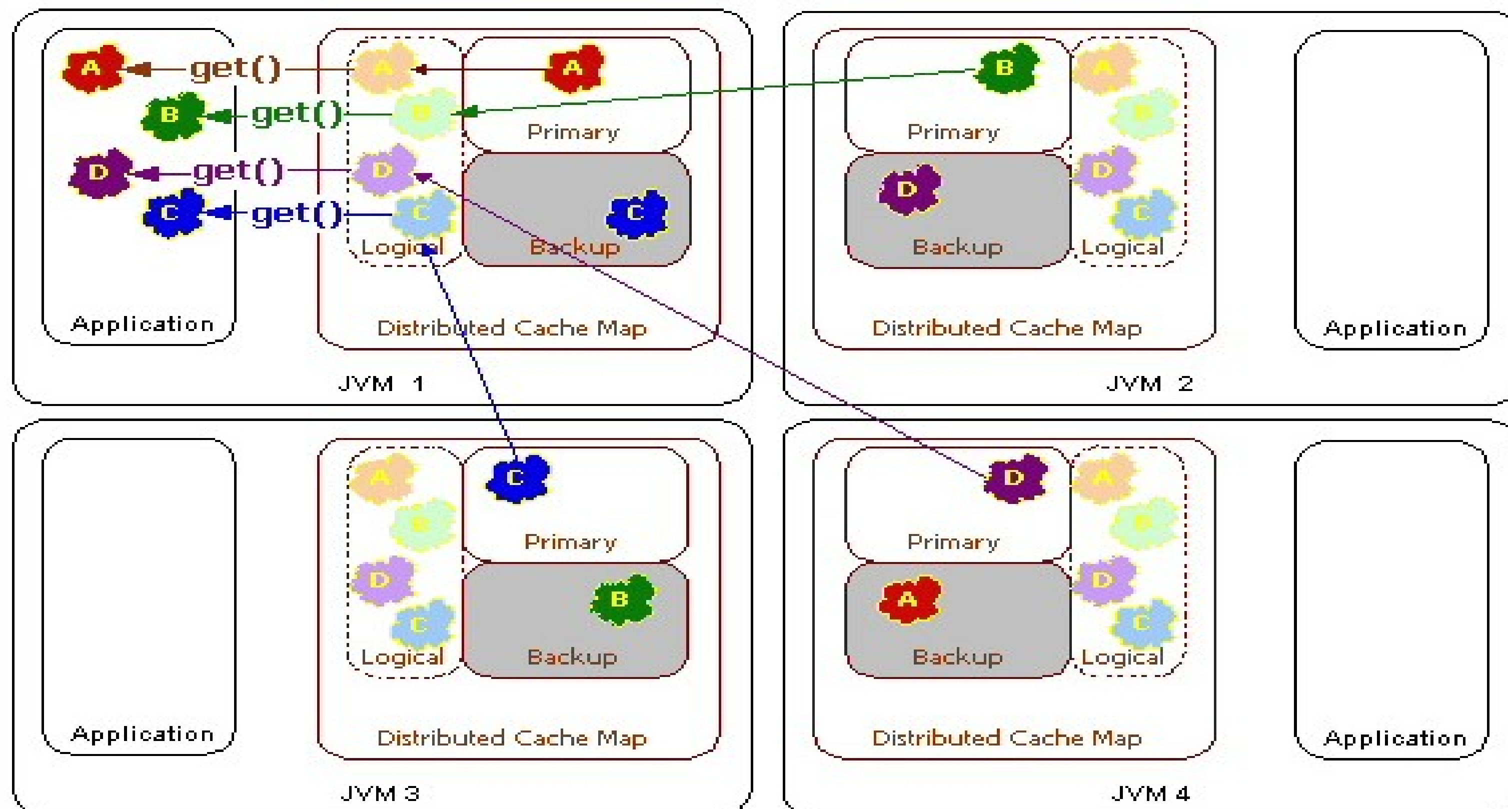
© Copyright Tangosol, Inc. 2005
All Rights Reserved

Partitioned Topology: Description

- ***Requirement*** **: Extreme Scalability.**
- ***Solution*** **: Shared-Nothing Architecture. Automatically Partition data across all cluster members.**
- ***Result*** **: Linear Scalability. By partitioning the data evenly, the per-port throughput (the maximum amount of work that can be performed by each server) remains constant as servers are added, up to the extent of the switched fabric.**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Partitioned Topology: Read Access



© Copyright Tangosol, Inc. 2005
All Rights Reserved

Partitioned Topology: Benefits

- ***Partitioned*** **: The size of the cache and the processing power available grow linearly with the size of the cluster.**
- ***Load-Balanced*** **: The responsibility for managing the data is automatically load-balanced across the cluster.**

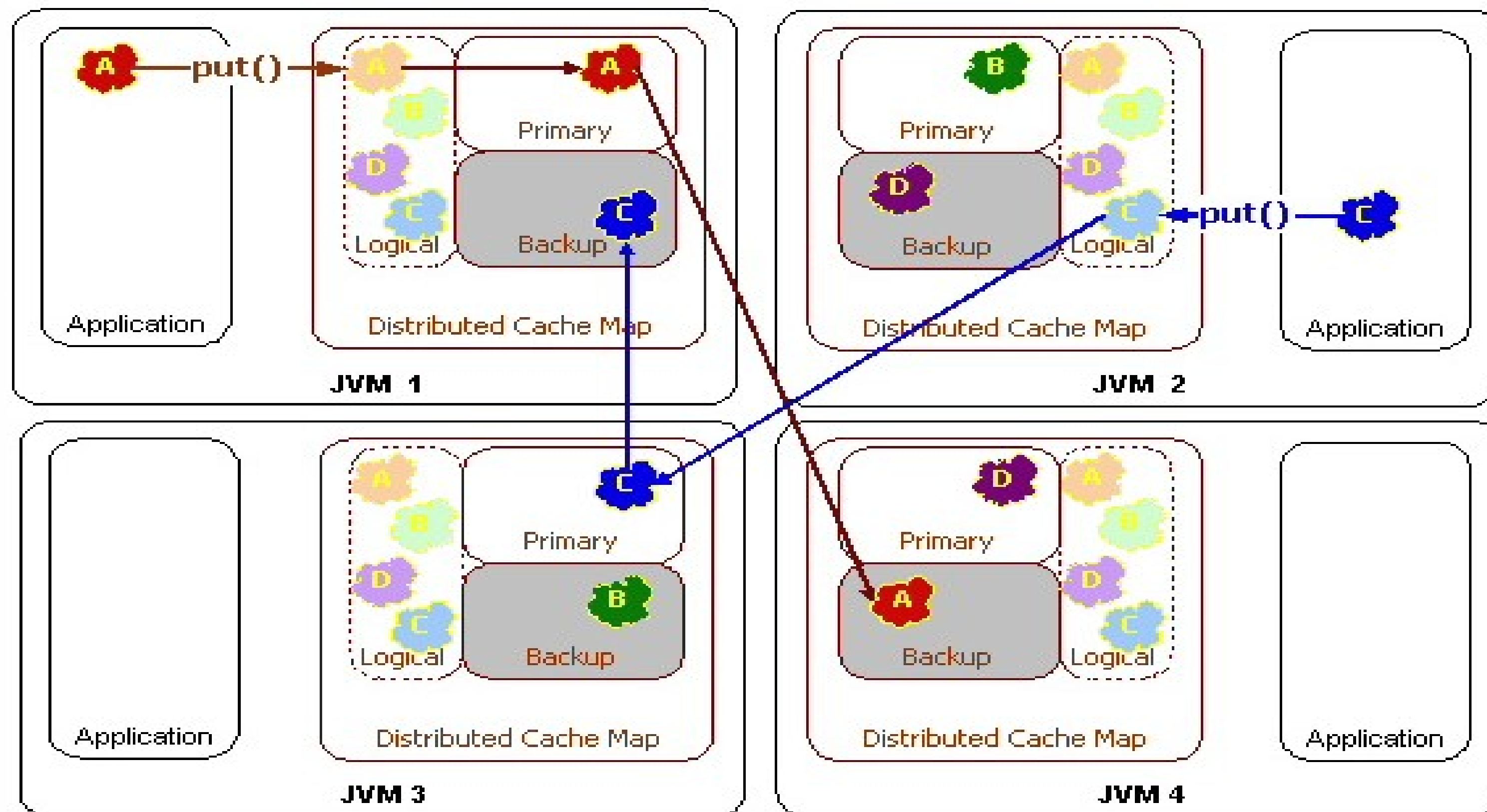
© Copyright Tangosol, Inc. 2005
All Rights Reserved

Partitioned Topology: Benefits

- ***Ownership*** **: Exactly one node in the cluster is responsible for each piece of data in the cache.**
  - **Supports cache-through architectures**
  - **Supports data-grid capabilities**
- ***Point-To-Point*** **: The communication for the Partitioned cache is all point-to-point, enabling linear scalability.**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Partitioned Topology: Write Operations



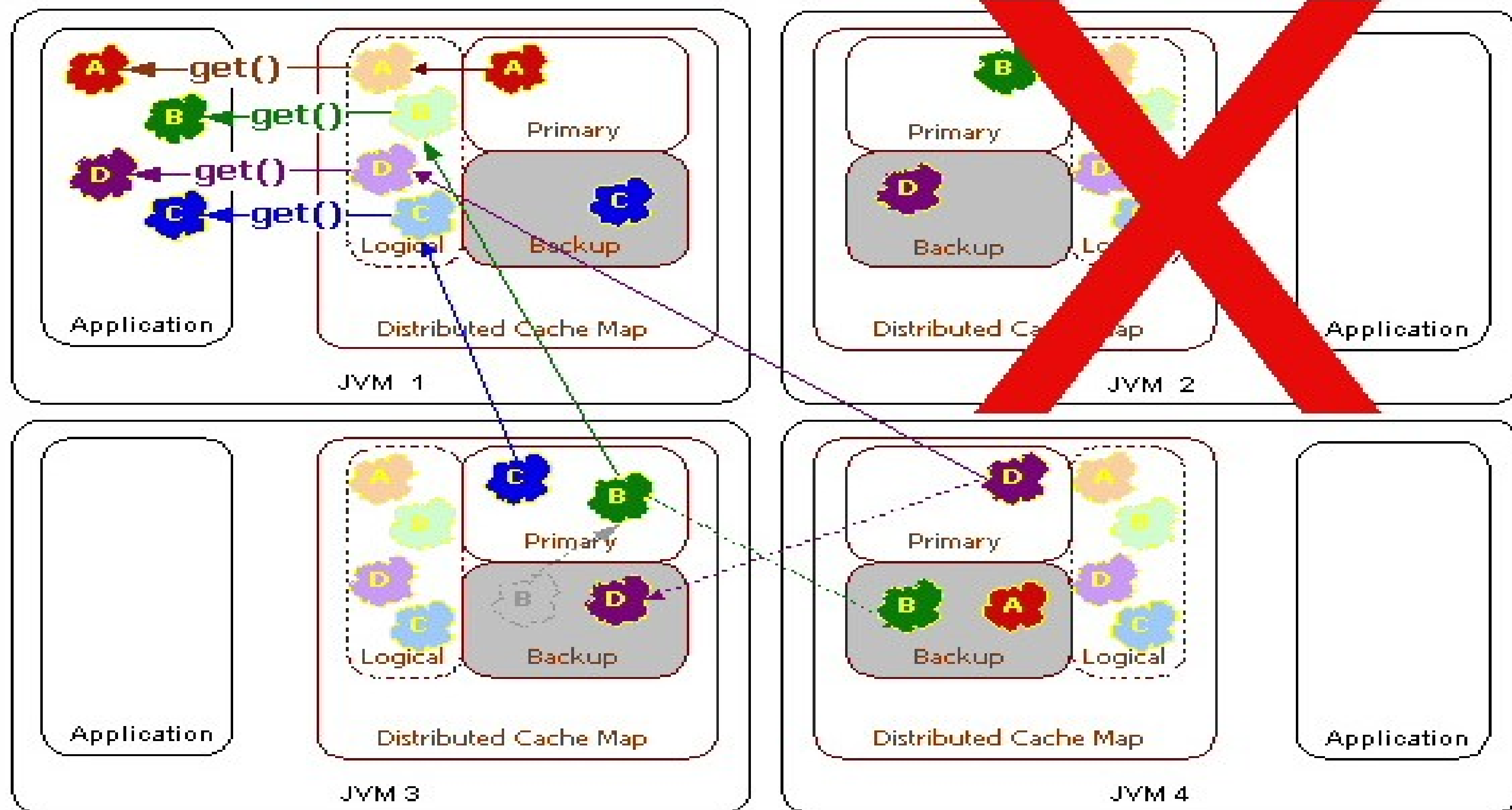
© Copyright Tangosol, Inc. 2005
All Rights Reserved

Partitioned Topology: Failover

- ***Failover*** **: All Coherence cache services provide failover and failback without any data loss, and that includes partitioned caches.**
  - **Configurable level of redundancy (backup count)**
  - **Any cluster node can fail without the loss of data**
  - **Data is explicitly backed up on different physical servers**
  - **Never a moment when the cluster is not ready for a server to die (no data vulnerability, no SPOF)**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Partitioned Topology: Failover



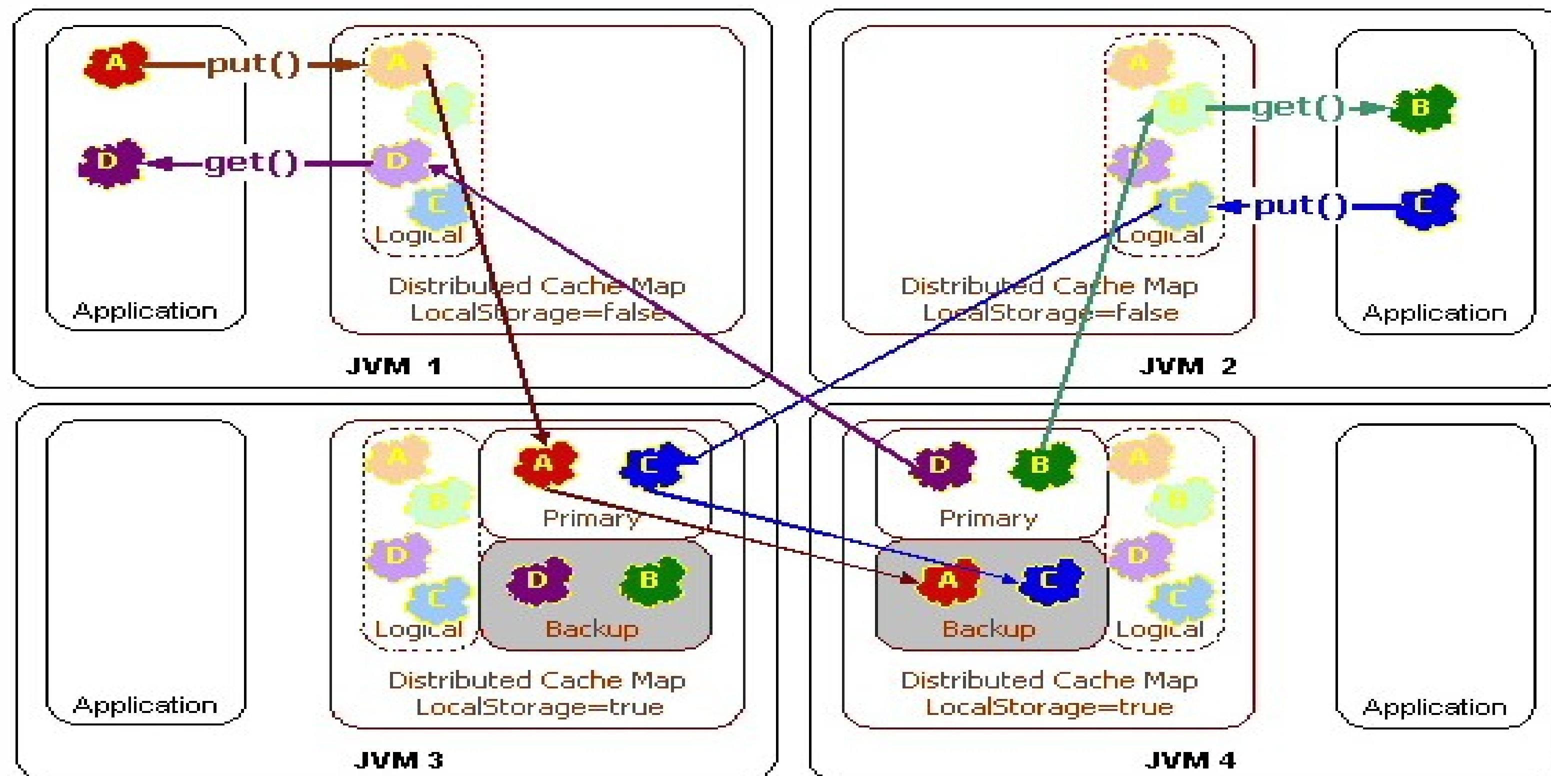
© Copyright Tangosol, Inc. 2005
All Rights Reserved

Partitioned Topology: Cache Servers

- ***Location Transparency*** **: The JCache API and its behavior are the same with a local, replicated or partitioned cache.**
- ***Local Storage Enabled*** **: Cluster nodes with local storage enabled will provide the cache and backup storage for the partitioned cache. All cluster nodes will have the same exact view of the data, because of Location Transparency.**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Partitioned Topology: Cache Servers



© Copyright Tangosol, Inc. 2005
All Rights Reserved

Partitioned Topology: Conclusions

- ***Performance*** **: Fixed cost.**
- ***Scalability*** **: Linear scalability of both cache capacity and throughput as the number of members increases. Designed for scaling on modern switched networks.**
- ***Uses*** **: Any size caches, scaling with the size of the cluster or data grid. Both read- and write-intensive use cases. Ability to offload heap usage to other JVMs. Load-balancing. Resilient to server failure.**

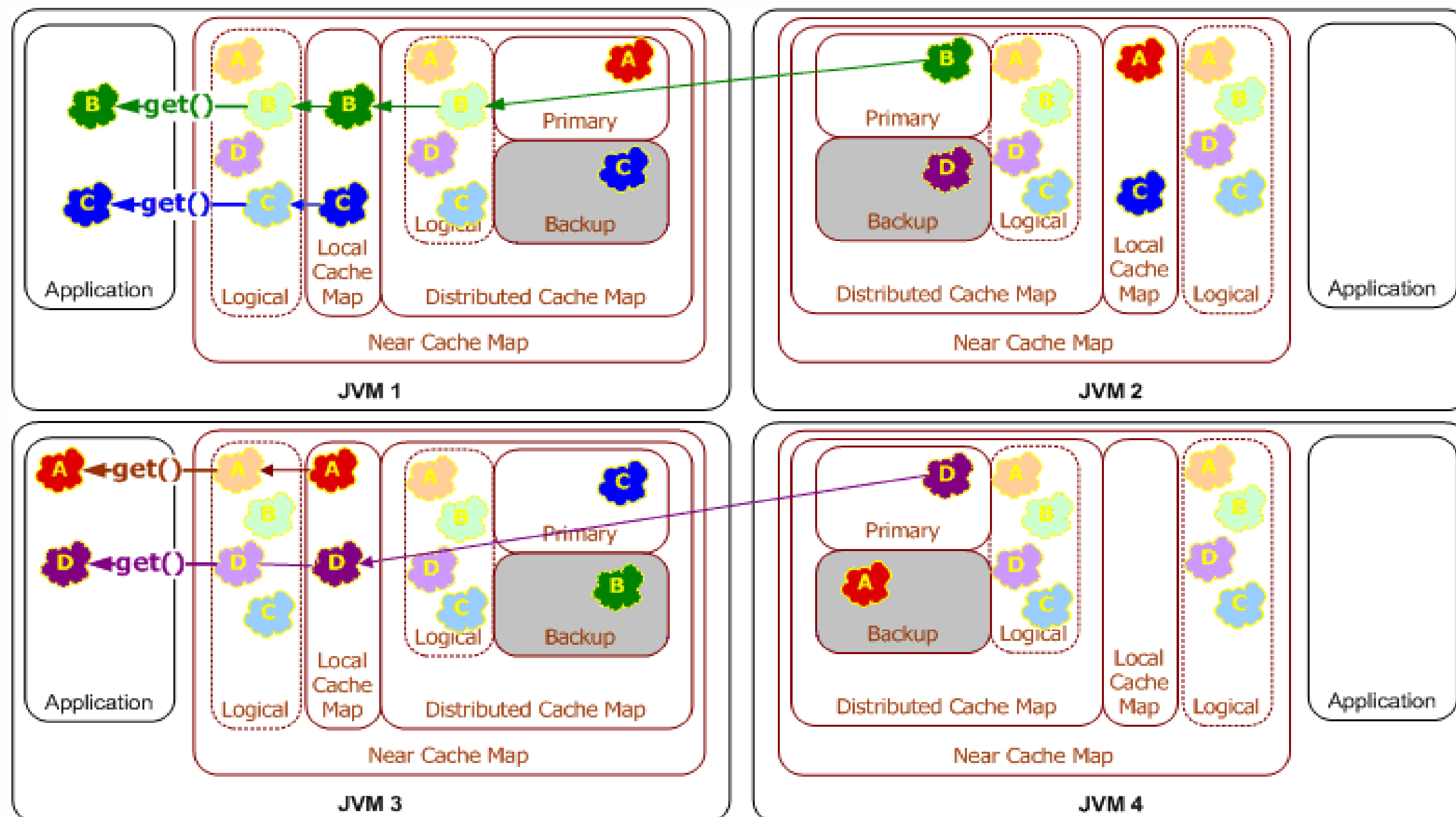
© Copyright Tangosol, Inc. 2005
All Rights Reserved

Near Topology: Description

- ***Requirement*** **: Extreme Performance. Extreme Scalability.**
- ***Solution*** **: Local "L1" In-Memory cache in front of a Clustered "L2" Partitioned Cache.**
- ***Result*** **: Zero Latency Access to recently-used and frequently-used data. Scalable cache capacity and cache throughput, with a fixed cost for worst-case.**

TANGOSOL
© Copyright Tangosol, Inc. 2005
All Rights Reserved
TechTarget
The Most Targeted IT Media

Near Topology: Read Access



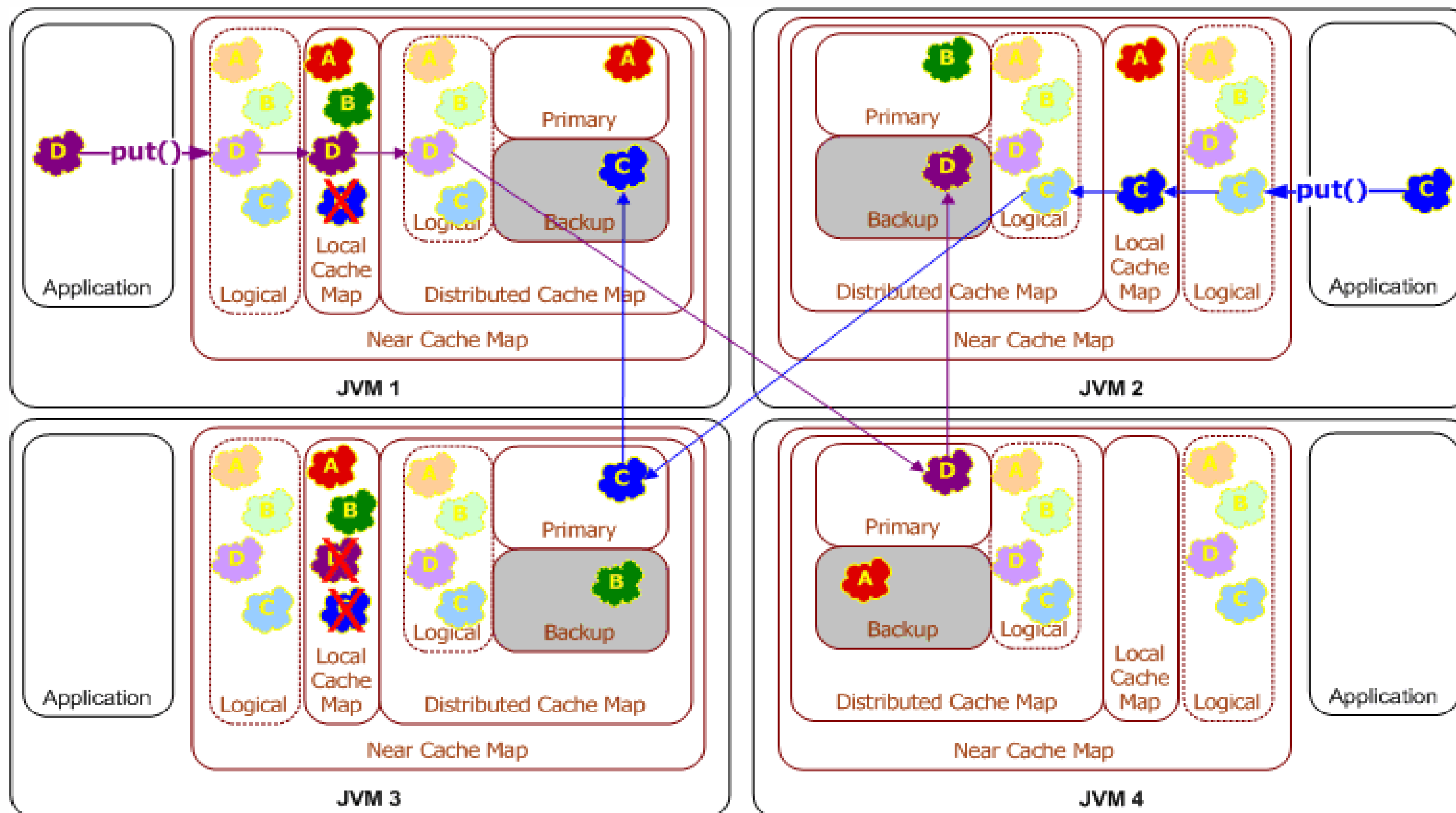
© Copyright Tangosol, Inc. 2005
All Rights Reserved

Near Topology: Coherency

- ***Read-Only/ Read-Mostly*** **: Expiry-based Near Caching allows the data to be read until its configured expiry**
- ***Event-Based Seppuku*** **: Eviction by event. The Near Cache can automatically listen to all cache events, or only those cache events that apply to the data it has cached locally.**

TANGOSOL
© Copyright Tangosol, Inc. 2005
All Rights Reserved
TechTarget The Most Targeted IT Media

Near Topology: Write Operations



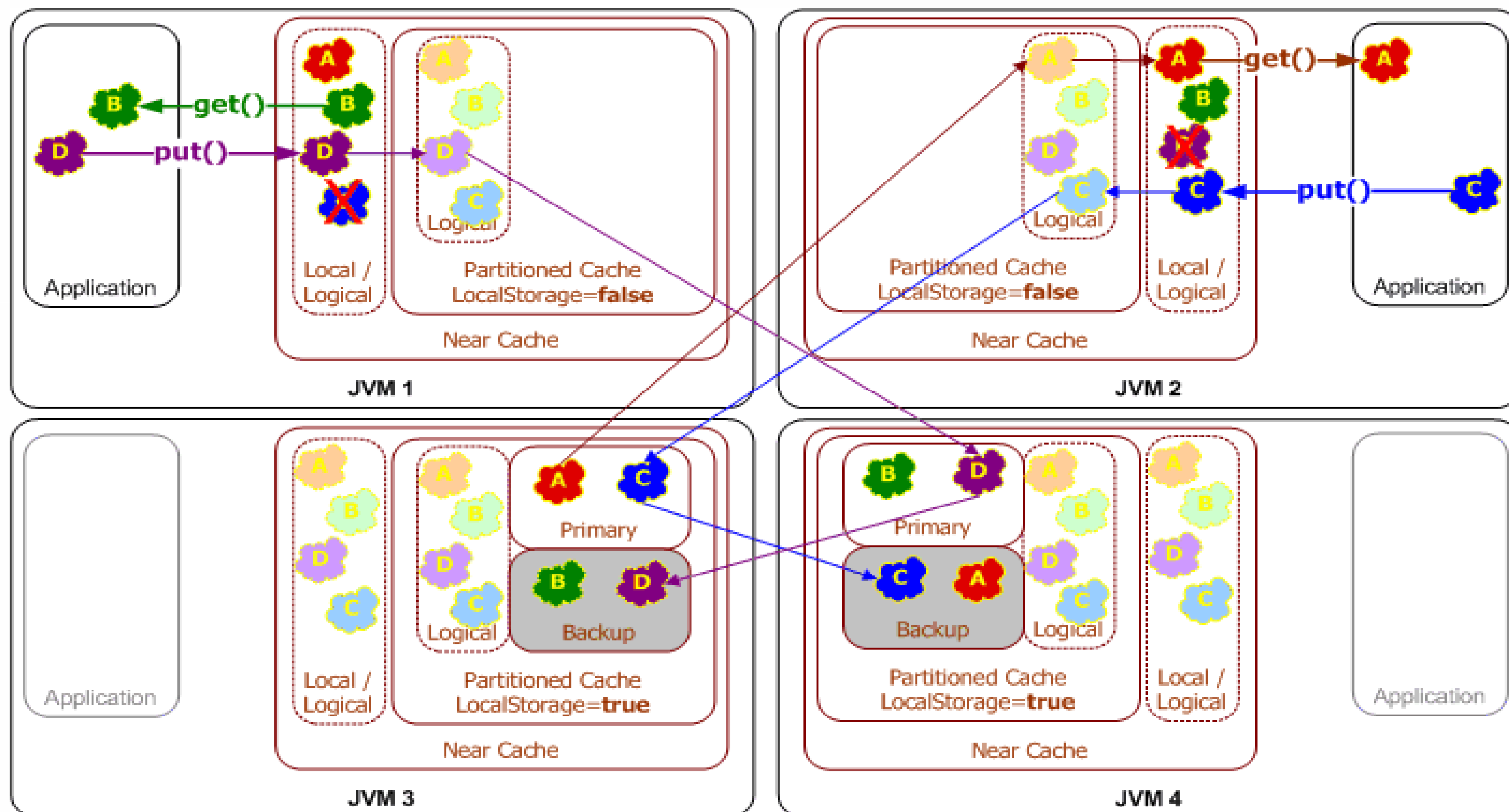
© Copyright Tangosol, Inc. 2005
All Rights Reserved

Near Topology: Cache Servers

- ***Potent Combination*** **: Combining the benefits of Near Caching with the dedicated Cache Servers can provide "the best of both worlds" for many common use cases.**
- ***Bulging At The Heap*** **: This topology is very popular for application server environments that want to cache very large data sets, but do not want to use the application server heap to do so.**
- ***Balanced*** **: The application server will use a tunable amount of memory to cache recently- and frequently-used objects in a local cache.**

TANGOSOL
© Copyright Tangosol, Inc. 2005
All Rights Reserved
TechTarget The Most Targeted IT Media

Near Topology: Cache Servers



© Copyright Tangosol, Inc. 2005
All Rights Reserved

Near Topology: Conclusions

- ***Performance*** **: Zero-latency for common data. Fixed cost for the remainder of the data.**
- ***Scalability*** **: Linear scalability of both cache capacity and throughput as the number of members increases. Some reduction in scalability when using event-based Seppuku.**
- ***Uses*** **: Any size caches, scaling with the size of the cluster. Both read- and write-intensive use cases. Particularly good for read-intensive caches that have tight data access patterns. Killer "Cache Server" config.**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Cache-Aside Architecture

- **Cache-Aside refers to an architecture in which the application developer manages the caching of data from a data source**
- **Adding cache-aside to an existing application:**
  - **Check the cache before reading from the data source**
  - **Put data into the cache after reading from the data source**
  - **Evict or update the cache when updating the data source**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Cache-Through: Architecture

- **Cache-Through places the cache between the client of the data source and the data source itself, requiring access to the data source to go through the cache.**
- **A Cache Loader represents access to a data source. When a cache is asked for data, if it is a cache miss, then any data that it cannot provide it will attempt to load by delegating to the Cache Loader.**
- **A Cache Store is an extension to Cache Loader that adds the set of operations generally referred to as Create, Read, Update and Delete (CRUD)**

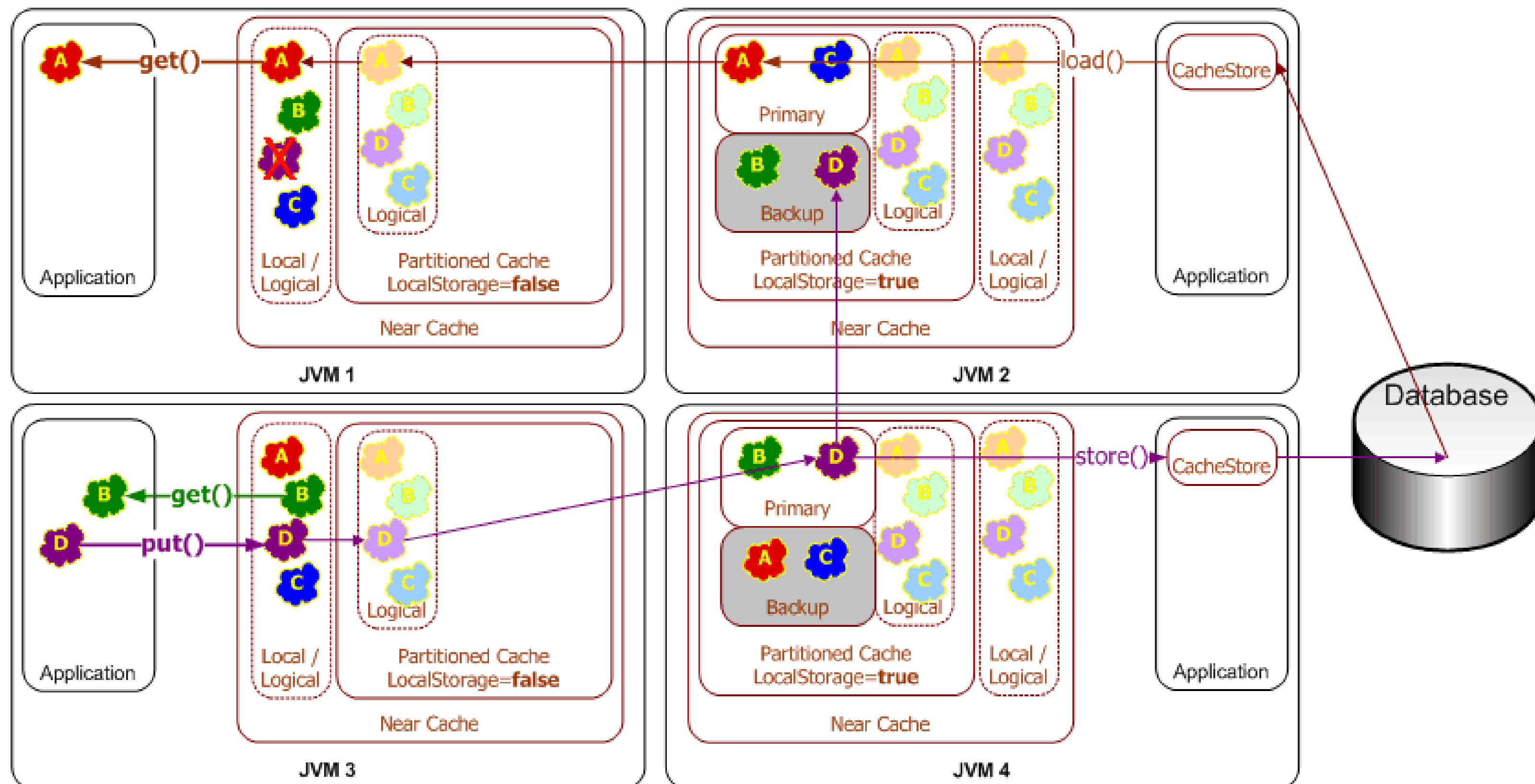
© Copyright Tangosol, Inc. 2005
All Rights Reserved

Cache-Through: Coherence

- **Coherence Cache-Through operations are always managed by the *owner* of the data within the cluster.**
- **Concurrent access operations are combined by the owner, greatly reducing database load.**
- **Write-Through keeps the cache and database in sync by keeping the cache aware of updates**

TANGOSOL
© Copyright Tangosol, Inc. 2005
All Rights Reserved
TechTarget
The Most Targeted IT Media

Cache-Through: Read- & Write-Through



© Copyright Tangosol, Inc. 2005
All Rights Reserved

Cache-Through: Conclusions

- ***Performance*** **: Reduces latency for database access by interposing a cache between the application and the data. Database modifications additionally involve a cache update.**
- ***Scalability*** **: Channels and combines data accesses. May significantly reduce database load.**
- ***Uses*** **: Any time a cache needs to transparently load data from a database. Clean encapsulation of data source access in one place: Cache Loader / Store.**

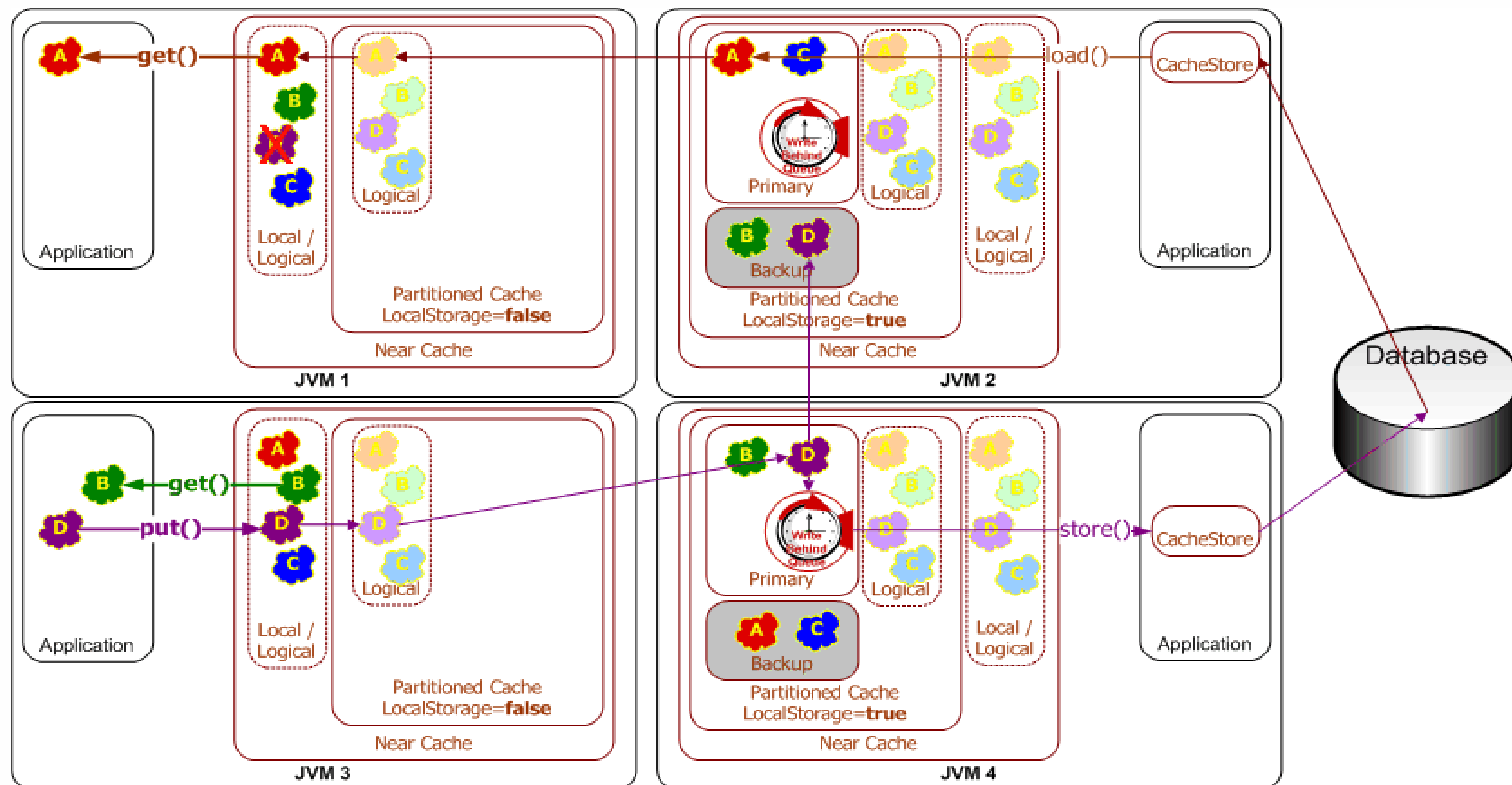
© Copyright Tangosol, Inc. 2005
All Rights Reserved

Write-Behind: Description

- Coherence Write-Behind accepts cache modifications directly into the cache
- The modifications are then asynchronously written through the Cache Store, optionally after a specified delay
- The write-behind data is clustered, making it resilient to server failure

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Write-Behind: Illustration



© Copyright Tangosol, Inc. 2005
All Rights Reserved

Write-Behind: Conclusions

- ***Performance*** **: Low-latency for cached data reads and data modifications.**
- ***Scalability*** **: The same extreme read/write scalability of the cache, and significantly (90%+) reduced load on the database**
  - **Coalesced write-through of multiple modifications to an object**
  - **Batched write-through of modifications to multiple objects**
- ***Uses*** **: When write performance is important, data source load is high, and/or when an application has to be able to continue when the data source is down.**
  - **Only use when all writes to the data source come through the cache**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Topology Quiz: Example Use Case

- **What cache topology would be optimal for this application?**
  - **Caching 10GB of financial portfolio data**
  - **Read-heavy, updated nightly**
  - **Several hundred users**
  - **Several thousand requests per minute**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Topology Quiz: Example Use Case

- **What cache topology would be optimal for this application?**
  - **Caching user preferences for an in-house application**
  - **Several hundred concurrent users**
  - **Preferences updated a few times per day**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Topology Quiz: Example Use Case

- **What cache topology would be optimal for this application?**
  - **Caching browsing history for an online retailer to support personalization and real-time assistance**
  - **Many updates, few reads**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Topology Quiz: Example Use Case

- **What cache topology would be optimal for this application?**
  - **Logging user interactions to a database for internal auditing purposes**
  - **1000 updates per minute**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 1: Use an MVC Architecture

- **Model/ View/ Controller (MVC, aka Model2)**
  - **Model: Domain-specific representation of the information the application displays and on which it operates**
  - **View: Renders the model into a form suitable for interaction, typically a user interface element or document**
  - **Controller: Responds to events – typically are user actions or service requests – and invokes changes on the model**

TANGOSOL
© Copyright Tangosol, Inc. 2005
All Rights Reserved
TechTarget The Most Targeted IT Media

Lesson 1: Use an MVC Architecture

- **Clear delineation of responsibility; for example:**
  - **Cache is used in the Model**
    - **Cache Loader / Store is the DAO**
    - **Cache contains the application's POJOs / Value Objects**
  - **View pulls data from the Model**
    - **Goal is to ensure that all accesses are served from cache**
  - **Controller affects the Model**
    - **Modifications via Write-Through or Write-Behind**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 2: Specify the Data Access

- **There are multiple ways to access data**
  - **ORM (JDO, EJB3, Hibernate, etc.)**
  - **Cache API (i.e. transparently via a Cache Loader)**
  - **JDBC (or other direct integration API)**
- **Most applications have a "best way"**
  - **Large-scale set oriented access usually indicates JDBC**
  - **Mix of set- and identity-oriented access indicates ORM**
  - **Identity-oriented access may indicate Cache API**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 2: Specify the Data Access

- **Picking the wrong approach is disastrous**
  - **RDBMS (JDBC) optimized for set-based queries and operations, including joins and aggregates, but crumbles with heavy row-level access (1+N access pattern, etc.)**
  - **ORMs can bog down bady on large set-based access**
  - **JCache API is built around identity-based access, not set-based access**

TANGOSOL
© Copyright Tangosol, Inc. 2005
All Rights Reserved
TechTarget
The Most Targeted IT Media

Lesson 2: Specify the Data Access

- **Not always an obvious "best choice"**
  - **Some applications have a mix of intensive row-level and large set-level operations, which lend themselves poorly to any single approach**
  - **Even a well-architected and carefully-designed application will often have a few "exceptions to the rule" that require the specified approach to Data Access to be circumvented**
  - **It is sometimes necessary to use different approaches for different classes of data within the same application**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 2: Specify the Data Access

- **Optimizations available for each**
  - **It is often possible to cache JDBC result sets**
  - **Most ORMs have effective support for pluggable caches**
  - **Some ORMs have optimizations for set-based operations that can translate some operations directly into optimized SQL that performs the entire operation within the RDBMS**
  - **Coherence provides extensive query support, including parallel query with indexes and cost-based optimizations**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 3: Design a Domain Model

- **Domain Model includes two aspects of application modeling**
  - **Data Model: Describes the state that the application maintains, both in terms of persistent data (e.g. the "system of record") and runtime data (sessions, queued events, requests, responses, etc.)**
  - **Behavioral Model: Describes the various actions that can affect the state of the application. Very similar to the concepts behind SOA, but at a much lower level.**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 3: Design a Domain Model

- **Domain Model is not dissimilar from SOA**
  - **Data Model: Analogous to the information encapsulated and managed behind a set of services**
  - **Behavioral Model: Analogous to the set of services exposed by a broker**
- **The concept of a Domain Model is technology-neutral. It can even exist only in the abstract.**
  - **Modeling is a tool, not a religion**

TANGOSOL
© Copyright Tangosol, Inc. 2005
All Rights Reserved
TechTarget
The Most Targeted IT Media

Lesson 3: Design a Domain Model

- **Domain Model has value**
  - **Allows the Data Model to exist independently of the behavioral model, supporting the separation of a controller from the model in an MVC architecture**
  - **The behavioral model is the basis for the events that a controller is required to support**
  - **With an abstract data model that reflects application concerns instead technology concerns, it is much more likely that the resulting data model implementation will be more easily used by the view and the controller**

TANGOSOL
© Copyright Tangosol, Inc. 2005
All Rights Reserved
TechTarget The Most Targeted IT Media

Lesson 3: Design a Domain Model

- **Domain Model is not new**
  - OO developers have been using Domain Modeling for years
  - Application Developers that double as DBAs have used modeling to "get their ideas down" into a design that could provide both optimal application implementation and optimal database organization
  - SOA is the publishing of a behavior model that is intended to be publicly-accessible, with data models often directly reflected in the service request and response data

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 4: Find the natural granularity

- **Every application has a natural granularity for its data.**
  - **Relational data models have a *normalized* granularity from which tables naturally emerge**
  - **Optimized JDBC-based applications have a statement execution granularity and a Result Set granularity**
  - **ORM-based applications and cache-intensive applications often have an OO granularity that mirrors the data model**
  - **Caches have an identity granularity of access**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 4: Find the natural granularity

- **Caches will typically exist for each major class of application object**
  - **e.g. Accounts, Symbols, Positions, Orders, Executions, etc.**
- **Each cache will tend to have a natural key**
  - **e.g. account id, symbol, account id + symbol, order id, etc.**
- **Application objects tend to be complex**
  - **Contain "owned" objects, e.g. Purchase Order contains Lines**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 5: Decouple using Identity

- Store, Load, Provide and Manage the Identity of related model objects
- Provide accessors for related model objects by using Identity de-reference (i.e. cache access)
- Read-only models tend to have more lee-way
  - Soft references
  - Transient reference fields

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 5: Decouple using Identity

- **Simplifies management of large object graphs**
- **Enables efficient lazy loading of object graphs**
- **Works well with …**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 6: Use an Immutable Data Model

- **From the View, the Model should treated as if it is read-only**
- **From the point of view of the Controller, the model that is shared across threads should be treated as immutable, for example just in case the View is using it on a different thread**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 6: Use an Immutable Data Model

- **Since most applications are *not* read-only, the Controller does have to modify the data represented by the Model**
- **When the Controller needs to modify the Model, it can obtain mutable clones of the shared model, and manage them …**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 7: Use Cache Transactions

- **… transactionally.**
- **When the Controller obtains cached values within a transaction, the values are actually clones of the "master" cached values**
- **The Controller makes its modifications to the Model in a transactionally isolated and consistent manner**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 7: Use Cache Transactions

- **For maximum scalability, most transactions should be optimistic. Just as with any optimistic transaction approach, this implies that the application must handle and/or retry transactions whose optimistic checks fail**
- **Cache Transactions can integrate with the container's Transaction Manager via the JEE Connector Architecture**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 8: Use Queries Wisely

- **Cache Queries are optimized, and they are run in parallel across a cluster, but they are probably at least an order of magnitude more expensive than identity-based operations**
- **If you use queries, make sure to use indexes; the Coherence query optimizer can use multiple indexes on a single query, even if they don't perfectly "cover" the query**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 9: Optimize Serialization

- **Objects that are stored in a cache may need to be serialized, and Java's default object serialization is relatively inefficient**
- **Implementing the Externalizable interface may help slightly**
- **Serialization using data streams instead of object streams can make a phenomenal impact**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 9: Optimize Serialization

- **Coherence provides the ExternalizableLite interface to support serialization to data streams, and the Tangosol XmlBean framework which implements ExternalizableLite for derived value objects**
- **Serialization performance improvements are up to an order of magnitude, and the reduction in size can be up to 80%.**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 9: Optimize Serialization

- **Since Java does not have an object-cloning interface, classes that do not have a public clone() method may require serialization and deserialization in order to be cloned**
- **Since Cache Transactions may need to clone an object to create a copy within a local transaction, optimized serialization can even improve the performance of transactions**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 10: Use Good Identities

- An Identity implementation must provide correct hashCode() and equals() implementations, and cache the hash-code!
- A good toString() implementation helps with debugging (and not just for Identities!)
- If feasible, make your Identity classes immutable

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 10: Use Good Identities

- An Identity must be Serializable, and its serialized form should be stable: Two instances should serialize to the same binary value if and only if equals() returns true
- Java's String, Integer, Long, etc. are perfect
- Tangosol XmlBean is a good base class for (or example of) complex identities

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 11: Cache in the Right Scope

- **HTTP Session objects can be used for caching user- or session-specific information; don't use them as a cache for global information**
- **Conversely, don't use a global cache for user-specific caching when the HTTP Session would do just fine**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Lesson 12: Never Assume it Works

- **We have seen caches in production that literally were not even getting used, and we have seen caches that had not even been configured – or were badly mis-configured**
- **While load testing, use JMX to monitor what caches exist, how big they are, what their hit rates are, and whether their average access times and access counts seem correct**

© Copyright Tangosol, Inc. 2005
All Rights Reserved

Audience Response

Question?

TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience.

Distributed Caching: Essential Lessons

Considerations for maximizing scalable performance and reliability in J2EE clusters

TANGOSOL

© Copyright Tangosol, Inc. 2005
All Rights Reserved

TechTarget
The Most Targeted IT Media