Typical Web Applications

Web Tier (ASP.Net)

Out of proc Session State Server

Data Tier

Database is hot
What is "Distributed Caching"?

• An explicit, distributed, in-memory application cache for all kinds of data (Java/.Net objects, rows, XML, Binary data etc.)
  – Fuse "memory" across machines into a unified cache
Where does it fit?

Web Tier (ASP.Net)

Cache Tier
# Distributed Cache Usage

<table>
<thead>
<tr>
<th>Verticals</th>
<th>Scenario</th>
</tr>
</thead>
</table>
| **Web**   | - User-specific HTTP session and shared state across web farm  
- In-flight shopping carts for web retail  
- Enabling online self-service applications  
- Explicit storage of pre-computed or highly-accessed data |
| **LOB**   | - Enterprise-wide product catalog for POS, analytics  
- Caching frequently used reference data for a ERP application |
| **Telco** | - Cellular/VOIP: compute utilization, prepay charges, call routing and session info  
- SMS: message content / notification / receipt, billing |
| **Travel** | - Aggregated flight pricing / availability retrieved from airlines |
| **Defense** | - Sensor network data processing and threat detection |
| **Financial** | - Per-user portfolio data and delayed quote storage for trading  
- Aggregate and process ticker stream for algorithmic trading |
# Types of Application Data

<table>
<thead>
<tr>
<th>Reference</th>
<th>Activity</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Read Only</td>
<td>Read-Write Not shared</td>
<td>Read-Write, Shared</td>
</tr>
<tr>
<td>Catalog Data</td>
<td>Shopping Cart</td>
<td>Auction Data/Seat Assignment</td>
</tr>
</tbody>
</table>

**Diagram:**
- Grocery Shop
- Web/App Tier
- Shopping Cart
- Grocery Catalog
- Distributed Cache
- Grocery Inventory
A version of the authoritative data
  - Aggregated or transformed
Each version is unique
Refreshed periodically
Examples
  - Web and Enterprise (Product) Catalogs
  - User, Employee data
Access pattern
  - Mostly read
  - Shared & Concurrent Access
Scale
  - Large number of accesses
Functionality
  - Key based Access
  - Simple Query & Filtering
  - Loading

Scenario: Social Networking
• Data typically generated as part of the application activity
  • Active during business transactions
    – Typically logged to a backend data source
    – Historical data
  • Examples
    – Shopping Cart
    – Session State
    – Enterprise LOB app (Purchase Order)
• Access pattern
  – Read and write
  – Primarily exclusive access
• Scale
  – High data (and access) scale
• Functionality
  – Key based access
  – Transactions (Grouping)
Caching Resource-oriented Data

- Authoritative data
- Modified by transactions; spans transactions
- Examples
  - Flight Inventory
- Access pattern:
  - Read and write
  - Shared access
- Functionality
  - Key based access
  - Transactions
- Scale
  - Large number of concurrent accesses
  - Relaxed consistency for scale

Scenario: Flight Inventory and Pricing

- Booking Service
- App Logic
- Distributed Cache
- Flight Routing Itinerary
- Flight Segment Flight Price
- Airlines
  - Continental
  - American Airlines
  - United
A version of the authoritative data
  – Aggregated or transformed
Several TBs on 100s of Memcached Servers
Examples
  – User data, friend data, pictures
Most accesses hit the cache
Access pattern
  – Mostly read
  – Shared & Concurrent Access
Scale
  – Large number of accesses
Functionality
  – Key based Access
  – Simple query/Filtering
Extreme Transaction Processing

- Distributed TP applications with exceptionally demanding performance, scalability, availability
- Real-time, business critical, secure, and manageable

- Traditional TP monitors
- Enterprise Application Servers
- Traditional Integration Brokers
- Message Servers

- Event Driven Messaging
- Enterprise/Internet Service Bus
- Grid/Fabric based Application Servers
- Low latency platform
Grid/Fabric based Application Servers

- Integrated distributed caching platform
- Application State Management
- Partitioned and Replicated application state

- Co-located logic and state
- Data aware routing
- Extreme low latency routing and access
- Durability and Persistence
Next generation applications – distributed, loosely-coupled, even-driven requiring high scale, performance and availability.
Application Requirements

- Efficient (Application) State management
- Performance
  - Millisecond/microsecond access
  - 100s of 1000s of accesses
- Scale
  - 10s – 100s of nodes in enterprise
  - 100s – 1000s in cloud applications
- Availability
  - Always available
- Consistency
  - Different degrees: Strong, Weak, Eventual, . . .
- Access
  - Key based and simple query based access
  - Transactions, Optimistic concurrency control
  - Invalidations
Caching API

// Create instance of cachefactory (reads appconfig)
DataCacheFactory fac = new DataCacheFactory();

// Get a named cache from the factory
DataCache catalog = fac.GetCache("catalogcache");

// Simple Get/Put
catalog.Put("toy-101", new Toy("Puzzle", .,.));

// From the same or a different client
Toy toyObj = (Toy)catalog.Get("toy-101");

// Region based Get/Put
catalog.CreateRegion("toyRegion");

// Both toy and toyparts are put in the same region
catalog.Put("toy-101", new Toy( .,.), “toyRegion”);
Catalog.Put("toypart-100", new ToyParts(…), “toyRegion”);

Toy toyObj = (Toy)catalog.Get("toy-101“, "toyRegion");
Access APIs – Tagging Items

- Add Tags to Items
  - Tag Search on Default Regions

```java
Tag hotItem = new Tag("hotItem");
catalog.Put("toy-101", new Toy("Puzzle"),
    new Tag[]{hotItem}, "toyRegion");
catalog.Put("toy-102", new Toy("Bridge"), "toyRegion");

// From the same or a different client
List<KeyValuePair<string, object>> toys =
    catalog.GetAnyMatchingTag("toyRegion", hotItem);
```
Usage Pattern – Cache Aside (Explicit Caching)

```csharp
// Read from Cache
Toy toyObj = (Toy)
catalog.Get("toy-101");

// If Not present in the cache
if (toyObj == null)
{
    // Read from backend..
    toyObj = ReadFromDatabase();

    // Populate Cache
    catalog.Put("toy-101", toyObj);

    return toyObj;
}
```
## Features

### API
- CRUD Operations (Create, Read, Update and Delete)
- Any Object type
- Multiple Client Languages
- Concurrency APIs
- Async and Batch APIs
- Transactions
- Query & Continuous Query
- Cache Notifications
- Eviction
- Persistence
- Session State (.NET, Java)
- IDE support

### Supported Topologies
- Partitioned
- Replicated
- Failover Support (High Availability)
- Multiple Backups
- Local Caching
- Explicit Data Affinity
- Embedded Cache
- Geo-replicated

### Extensibility
- Custom Eviction
- Custom Persistence
- Custom Query
- Triggers

### Persistence
- Read Through
- Refresh Ahead
- Write Through
- Write Behind

### Other
- Administration & Monitoring
- Security
- Co-location of logic & data in cache
# IMDB vs. Distributed Caching Platforms (DCPs)

<table>
<thead>
<tr>
<th>IMDB</th>
<th>DCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primarily relational store</td>
<td>Object store – any form of object</td>
</tr>
<tr>
<td>DB-specific representation</td>
<td>Application-specific representation</td>
</tr>
<tr>
<td>Only SQL query</td>
<td>Object/relational query (e.g. Linq, SQL)</td>
</tr>
<tr>
<td>Set-oriented access</td>
<td>Key based, Navigational, set-oriented access (e.g. GET, PUT, simple query)</td>
</tr>
<tr>
<td>Centralized</td>
<td>Distributed</td>
</tr>
<tr>
<td>Performance acceleration</td>
<td>Performance, Scale, and Failover</td>
</tr>
<tr>
<td>Server deployments</td>
<td>Embedded or server deployments</td>
</tr>
<tr>
<td>Niche, vertical markets (e.g. Telco)</td>
<td>General purpose (e.g. Web, LOB)</td>
</tr>
<tr>
<td>e.g. TimesTen, Solid DB, ANTS</td>
<td>e.g. memcacheD, Gemstone, Oracle Coherence, Gigaspaces, IBM extremeScale, AppFabric Caching etc.</td>
</tr>
</tbody>
</table>
DCP Players

• Memcached (open source)

• VMWare (Gemstone) Gemfire

• Gigaspaces Extreme Application Platform

• IBM WebSphere Extreme Scale Cache

• Microsoft AppFabric Caching

• Oracle Coherence

• Terracotta's Terracotta Server (open source)
Distributed Caching Platform Concepts
AppFabric Caching Logical Hierarchy

Host
- Physical processes hosting AppFabric Caching instance.

Named Caches
- Can span across machines
- Defined in the configuration file

Cache Item
- Key, Payload (Object), Tags, TTL, Timestamps, Version

Regions
- Physically co-located Container of Cache Items
- May be implicit or explicitly created

Machine -> Cache Host -> Named Caches -> Regions -> Cache Items -> Objects
Scale: Partitioned Cache

- **Application**
  - (K2, V2)
  - PUT
  - Cache Client1
    - Routing Table
    - Primary for K1, V1
      - K1, V1
    - Cache1
      - Routing Table
    - Primary for K2, V2
      - K2, V2
    - Cache2
      - Routing Table
      - Cache Client2
        - Routing Table
        - Cache Client3
          - Routing Table
    - Primary for K3, V3
      - K3, V3
    - Cache3
      - Routing Table

- **Using the Routing table client routes the PUT to cache2 (primary) node**
  - Get(K2)

- **Operations queue for notifications, to bring up a new secondary, etc.**
### Key Mapping

#### Region (Name)

<table>
<thead>
<tr>
<th>Region</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Region 1</td>
<td></td>
</tr>
<tr>
<td>Default Region 2</td>
<td></td>
</tr>
<tr>
<td>Default Region 256</td>
<td></td>
</tr>
<tr>
<td>ToyRegion</td>
<td>&quot;Toy101&quot;</td>
</tr>
<tr>
<td>ToyRegion</td>
<td>&quot;Toy102&quot;</td>
</tr>
<tr>
<td>&quot;BoxRegion&quot;</td>
<td>&quot;Box101&quot;</td>
</tr>
<tr>
<td>&quot;Cust1&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Cust2&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Cust33&quot;</td>
<td></td>
</tr>
</tbody>
</table>

#### Partition (Range of Ids)

- 0 – 1000
- 1001 - 2000
- \(\ldots\)
- \(\ldots\)
- \(\ldots\)
- \(\ldots\)
- \(\ldots\)
- \(\ldots\)
- \(\ldots\)
- XXX - Maxint

#### Region Name Hashed into Region Id

- ID Ranges mapped to Nodes
- Cache Service
Scale: Replicated Cache (Synchronous)

Application

Cache Client1

Routing Table

Using the Routing table client routes the PUT to cache2 (primary) node

• Queues the PUT operation
• PUTs locally
• Propagates operation to Cache1 and Cache3
• Returns control

Get(K2)

Cache1
Routing layer
Primary for (K1,V1)

Cache2
Primary for (K2,V2)

Cache3
Primary for (K3,V3)

Replication Agent

K1, V1
K2, V2
K3, V3

PUT

Queueing PUT...
Scale: Replicated Cache (Async)

Application

(K2, V2)

Cache2

Primary for (K2, V2)

Cache1

Routing Table

Cache Client1

PUT

Cache2 (primary) node

Using the Routing table client routes the PUT to cache2 (primary) node

Routing layer

Cache1

Primary for (K1, V1)

Cache2

Primary for (K2, V2)

Cache3

Primary for (K3, V3)

Using the Routing table client routes the PUT to cache2 (primary) node

• Queues the PUT operation
• PUTs locally
• Returns control
• Propagates operation to Cache1 and Cache3

K2, V2

K1, V1

K3, V3

K2, V2

K1, V1

K3, V3

Get(K2)

K2, V2

K1, V1

K3, V3

K2, V2

K1, V1

K3, V3
Local Cache

- Local Cache can help speed up access on clients
- Uses notification mechanism to refresh the cache on cache item changes

- **Put(K2, v2)**
- **Get(K2)**
- **Get(K2)**
Using the Routing table client routes the PUT to cache2 (primary) node

- Queues the PUT operation
- PUTs locally
- Propagates operation to secondaries (cache1 & cache3)
- Waits for a quorum of acks
- Returns control

Get(K2)

Cache Client
Failover

PM analyzes the info on secondaries of all primary partitions of Cache2 to elect the primaries.

Picks Cache1 as the primary for (K2,V2). Sends messages to the secondary caches, Cache1 and Cache3. Updates GPM

Detects Cache 2 failure. Notifies PM (on Cache4)

Cache1 initiates reconfiguration. After reconfig, Cache1 is primary for (K1, V1) and (K2, V2)

Cache1 polls secondaries (Cache2) to ensure it has the latest data; otherwise, it will give up primary ownership

Routing Table
Reconfiguration Agent
Replication Agent
Local Partition Map

Routing Table
Reconfiguration Agent
Replication Agent
Local Partition Map

Routing Table
Reconfiguration Agent
Replication Agent
Local Partition Map

Routing Table
Reconfiguration Agent
Replication Agent
Local Partition Map

Routing Table
Reconfiguration Agent
Replication Agent
Local Partition Map
Embedded Cache

- Cache client and server components run as part of the application process
- Avoids serialization and network costs
- Provides high performance, low latency access
- Guaranteeing locality and load balancing is tricky
- Better suited for replicated caches
Optimistic Version-based Locking

- GetCacheItem returns a version object
- Every update to an object internally increments its version
- Supply the version obtained along with the Put/Remove
- Put/Remove will succeed only if the passed in version matches the version in the cache

<table>
<thead>
<tr>
<th>Time</th>
<th>Client1</th>
<th>Client2 (Different Thread or process)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>Cachetle item = catalog.GetCacheItem(&quot;PlayerRegion&quot;, &quot;Zune&quot;);</td>
<td>Cachetle item = catalog.GetCacheItem(&quot;PlayerRegion&quot;, &quot;Zune&quot;);</td>
</tr>
<tr>
<td>T1</td>
<td>((ZuneObject)item.Object).inventory --;</td>
<td>((ZuneObject)item.Object).inventory--;</td>
</tr>
<tr>
<td>T2</td>
<td>catalog.Put(&quot;PlayerRegion&quot;, &quot;Zune&quot;, item.Object, item.Version);</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>catalog.Put(&quot;PlayerRegion&quot;, &quot;Zune&quot;, item.Object, item.Version);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>// Version mismatch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>// Client must retry again</td>
<td></td>
</tr>
</tbody>
</table>

Two clients access the same item
Both update the item
Second Client gets in first; put succeeds because item version matches; atomically increments the version
First client tries put; Fails because the versions don’t match
Pessimistic Locking

- Take locks on non-existent keys
- Allows you to co-ordinate creating new object amongst multiple clients
Scalable Notifications

Application

Caching Client

Routing Table

Partition: P2
Last LSN: 19

Call Delegate
Store Last LSN

Register Notification for Key “K3”

Map Keys to Partition (say P2)

Poll Required Nodes

Nodes Return List of Changes LSN Order

Cache1
Primary for
K1, V1
Change Log
33 Add K1
34 Del K22

Cache2
Primary for
K2, V2
Change Log
Partition P1
1 Add K2
2 Del K32

Cache3
Primary for
K3, V3
Change Log
(Partition P2)
18 Del K32
19 Del K43

Change Log
33 Add K1
34 Del K22
Eviction

• Expiry only eviction which
  – Evicts expired items alone
  – Periodic
  – Per partition

• Hard-eviction (Data > Allocated Cache Size)
  – Evicts expired items + non-expired items (in LRU order)
  – Per request
  – Can be turned off

• Memory pressure based eviction
  – A thread for detecting memory pressure (polling per second)
  – Avoids paging
  – Triggers hard-eviction (mentioned above) at 85% system memory usage and asks for releasing 5% of system memory
Persistence – Cache Through

- Callback for read-through, write-through, write-behind
- Specified at Named Cache Level
- Read-Through
  - Called when item not present in cache
  - Callback returns the object/serialized bytes
- Write-Through
  - Called when item is put
- Write-Behind
  - Writes to cache are queued
  - Callback called asynchronously in batches
  - Re-tries upon failure
- Bulk Access APIs
Read-Through Cache

Application

Get(K2)

Cache Client2

Routing Table

Cache2

Routing Table

Primary for K2, V2

K2, V2

Cache1

Routing Table

Primary for K1, V1

K1, V1

Cache3

Routing Table

Primary for K3, V3

K3, V3

DB
Write-Through Cache
Async Write-Back Cache

- Application
  - Put (K2, V2)
  - Cache Client2
    - Routing Table
- Cache1
  - Routing Table
  - Primary for K1, V1
  - K1, V1
- Cache2
  - Routing Table
  - Primary for K2, V2
  - K2, V2
- Cache3
  - Routing Table
  - Primary for K3, V3
  - K3, V3
- DB
Async Write Back (Write Behind) Cache

- Specified at Named Cache Level

- Write-Back
  - Asynchronously written to disk (e.g. database)
  - Physical write done via callbacks
  - Writes to cache are queued
  - Callback called asynchronously in batches
  - Re-tries upon failure
Executing A Query

from toy in catalog<Toy>()
where toy.ToyPrice > 300
select toy;

from toy in catalog<Toy>()
where toy.ToyPrice > 300
select toy;
Executing A Query

```csharp
from toy in catalog.GetRegion<Toy>("ToyRegion")
where toy.ToyPrice > 300
select toy;
```
DCP Architecture
Microsoft’s AppFabric Caching Architecture

Local Store Components
- In-memory Data Manager
  - Hash, B-trees
- Local Store Components
  - Object Manager
    - Query Processor
  - Policy Management
  - Notification Management
  - Region Management
- Distributed Object Manager
  - Dispatch Manager
  - Distributed Manager

Distributed Components
- Cluster Substrate
  - Failure Detection
  - Raw Transport
  - Reliable Routing
- Common Availability Substrate
  - Replication Agent
  - Reconfiguration Agent
  - Local Partition Map
  - Routing Table

Administration and Monitoring
- Tools Integration
- Cache Monitors

Client Layer
- Cache API
- Local Cache
- Federated Query Processor
- Dispatch Manager
- Routing Table

Administration and Monitoring
- Cache Monitors

Tools Integration
- Integration

In-memory Data Manager
- DM API
Customer & Usage Trends
Cache in Multi-tiered Application

Web Tier

Web1 → Web2 → Web3 → Web4 → Web5

Csh1 → Csh2 → Csh3

Que1 → Que2 → Que3

Application Logic

WF1 → WF2 → WF3

Csh1 → Csh2 → Csh3

Data Tier

DB1 → DB2 → DB3
Cloud Applications and Caching

• Application (and cache) on-premises and Data on Cloud

• Application and Data on Cloud
  – Cache as a service
  – Cache co-located with App

• Application on Cloud and Data on-premises
App on-premises; Data on Cloud

- ASP.Net Web Tier
- Application & Caching deployed On-premise
- Data on SQL Cloud
App on Cloud; Data on Cloud; Cache on a VM

Application
Caching Access Layer
Caching Server

Web servers
Application & Caching on Cloud
Caching VM
Data on Cloud
App on Cloud; Data on Cloud; Cache as a Service
App on Cloud; Data on-premise

Web servers
Application & Caching on Cloud
Caching VM
Data on-premises
Cloud – On-premises Connectivity

Application
Caching Access Layer
Caching Service

Application
Caching Access Layer
Caching Service

Application
Caching Access Layer
Caching Service

ASP.net

ASP.net

ASP.net
DCP Vendors

• Memcached (open source)
• VMWare (Gemstone) Gemfire
• Gigaspaces Extreme Application Platform
• IBM WebSphere Extreme Scale Cache
• Microsoft AppFabric Caching
• Oracle Coherence
• Terracotta's Terracotta Server (open source)
Distributed Caching Hard Problems

- Large caches
- Extreme Low Latency
- Impact of NVRAM technologies
  - PCM?
- Cache as the Truth?
- Durability?, Persistence?
- DBMS Capabilities?
Q/A?