Cloud Computing

WebSphere eXtreme Scale
Dan O’Riordan
IDR La Gaude, France, IIC Architects
Agenda

- Introduction
- Overview
- Key Capabilities and Use-Cases
- Essential WXS Architectural concepts
- Basic Implementation Scenarios
- Customer Success Stories
- The Competition
- Q&A
What is WebSphere XD?

Software to virtualize, control, and turbo-charge your application infrastructure

Infrastructure Optimization
Intelligent Workload Management
Virtualization

Automatic Sense & Respond Management

Data Fabrics & Caching

Batch Processing & Grid Computing
WebSphere XD Packaging Structure
Available as a single, integrated package or by 3 individual components

WebSphere Virtual Enterprise
(aka Operations Optimization)

WebSphere eXtreme Scale
(aka Data Grid)

WebSphere XD Compute Grid
(for now...)
High-Level Product Packaging Strategy

Split XD into 3 separate products that address distinct market opportunities and audiences.
The WebSphere Application Infrastructure Family

- WebSphere Virtual Enterprise
- WebSphere eXtreme Scale
- WebSphere Extended Deployment Compute Grid
- WebSphere Application Server for z/OS
- WebSphere Application Server Network Deployment
- WebSphere Application Server
- WebSphere Application Server Express
- WebSphere Application Server Community Edition

Customer Needs
- Ultimate scalability & performance; functional depth & breadth
- Reduced acquisition costs; Small footprint

Capabilities
- Fast deployment
- Get started quickly – Developers / Architects
- Free download

High Qualities of Service

Built on common WebSphere code
Built on open source technology
Gartner and XTP

Gartner defines XTP
"an application style aimed at supporting the design, development, deployment, management and maintenance of distributed TP applications characterized by exceptionally demanding performance, scalability, availability, security, manageability and dependability requirements."

Gartner RAS Core Research Note G00131036
WXS - IBM's DataGrid/XTP platform

- XD DataGrid/ObjectGrid is now renamed/relaunched as WebSphere eXtreme Scale
- Data Grids are a new technology being adopted by customers.

- It virtualizes free memory on a grid of Java virtual Machines (JVMs) into a single logical space and makes it accessible as a partitioned key addressable space for use by applications.

- It can make the stored data fault tolerant using memory replication policies.

- The space can be scaled out by adding more JVMs while it's running without restarting.

- It offers predictable **SCALING** and scaling at predictable **COST**.
Data Fabrics and Caching

Move beyond caching with WebSphere eXtreme Scale to develop game-changing and auto-scalable applications

ObjectGrid

Infrastructure for building extreme scale distributed systems:
- Automatic scaling
- Highly available
- Built in redundancy

Partitioning Facility

Supports a partitioned, asymmetric clustering model for highly scalable and reliable transaction processing
Lessons in Scalability
Caching and Scalability
Traditional Cache Operation

- Traditional in JVM cache
- Cache capacity determined by individual JVM Size.
- Invalidation load per server increases as cluster grows.
- Cold start servers hit the EIS even when data is cached in cluster.
- Lower performance as load increases due to invalidation chatter
- No redundancy of cached data
WXS based Cache Operation

- Cluster Coherent cache
- Cache capacity determined by cluster size, not individual JVM Size
- No invalidation chatter
- Cache request handling handled by entire cluster and is linearly scalable
- Load on EIS is lower
- No cold start EIS spikes
- Predictable performance as load increases
- Cached data can be stored redundantly

Cache cluster can be co-located with the application or run in it’s own tier.
WebSphere eXtreme Scale Entry Points

A flexible framework for realizing high performance, scalable and data-intensive applications

It can be used as a very powerful cache that scales from simple in-process topologies to powerful distributed topologies.

It can be used as a form of in memory database to manage application state (and it scales to 1000’s of servers). This is sometimes referred to as Distributed Application State Management.

It can be used as a platform for building powerful XTP/Data Grid applications.
A tiered configuration to offload backend processing or speed access to results…

Clients can use the distributed locking services provided by the remote ObjectGrid to coordinate access to shared data across clients.

Client Application

Local Cache

Very Large Cache

Data Tier

Caches all of the relevant data

Client caches a subset of the data

1..n clients
Horizontal Scalability via Auto-balancing of Data

With XD Operations Optimization, if more servers are added to the grid, then partition fragments from existing servers are transparently moved to the new servers to spread load in an even fashion.
High Availability and Fault Tolerance

If a primary server fails, then a replica is promoted to primary and ObjectGrid creates additional replicas according to the QoS desired. This is done automatically. Replicas can be either synchronous or asynchronous replicas.
Benefits of WXS

For the Executive
- State of the art cache technology
- Reduced Total Cost Of Ownership (TCO)
  - Automatic management of the growth
  - Continuous availability of application
  - Optimal use of back-end database
  - Resilient, fault tolerant infrastructure
- Coherent IT Strategy for data

For Administrator/Operations
- Tight integration with all WebSphere ND versions from V6.0
- Ability to run independently in a J2SE environment
- Self healing and self managing cluster management
  - Autonomic placement and rebalancing of data
  - Autonomic recovery of primaries and replica’s on catastrophic failure
- Flexible policy based placement of grid instances

For the Architect
- Need to sell WXS to the architects
- Coherent IT strategy for data for easy maintainability
- Easily extendable without code changes
- Allows for growing transaction volume with low latency; Linear scalability, as capacity is increased
- Seamless integration of cache into existing deployment, without coding effort
- Easy integration with other WebSphere products i.e. Commerce, Portal, Message Broker
- Use of standards where appropriate
  - Pluggable integration with JPA and Hibernate

For the Developer
- State of the art programming model using EntityManager for transparent POJO persistence
- Object graphs directly supported using the EntityManager APIs.
- Powerful query engine
- XML based configuration
Features of WebSphere eXtreme Scale

• In-memory database framework
  – By default, does not use disk as persistent storage mechanism

• Transactional
  – Ensures that grouped operations all succeed, or are all rolled back

• Highly available
  – Supports synchronous and asynchronous data replication

• Scalable
  – Provides horizontal scalability by distributing read requests
  – Provides horizontal scalability using data partitioning
  – Provides vertical scalability through efficient network IO and thread management
Features of WebSphere eXtreme Scale

- Secure
  - Supports Java and J2EE security standards

- Portable
  - Works in J2EE and J2SE environments

- Supports caching functions
  - Size management, invalidation, loading

- Customizable
  - Provides plug-in points to allow application specific behaviors

- Supports high-performance applications
  - API provides numerous ways to improve application performance
Why use WebSphere eXtreme Scale?

• It implements persistence using policy driven memory based replication and policy driven data placement
  – ObjectGrid stores transactional data in exactly one place and supports N copies of various quality levels.
  – Reference data can be replicated throughout the grid.
  – Business logic can run against copies of the data for additional scalability.

• It provides an alternative architecture for very high performance applications
  – Provides constant millisecond level responses as the system scales

• It can improve performance by
  – Caching data closer to the application
  – Reducing the database load
Why use WebSphere eXtreme Scale?

• It can lower the cost of a solution by reducing the database load

• It supports grid-based data centric transaction processing
  – The full power of the grid can be used to process large quantities of data at memory speeds with little change in response as the grid grows.
What are the major use cases of WXS?

- As an in-memory cache
- As a shared coherent cache
- As a database “shock absorber”
- As a multi-platform cache / data source aggregator
- As a scalable, low latency, highly available, data store
- As an HTTP session persistence mechanism
As an in-memory cache

- Application requirements
  - Application does not share data with other applications
  - Application does not have availability or scalability requirements for the data

- ObjectGrid function
  - ObjectGrid used as an in-memory cache for transient data
  - Multiple threads can access data transactionally
  - Queries against the data can be performed
  - Data can be indexed
As a Shared Coherent Cache

- Application requirements
  - Data shared between multiple applications, or multiple instances of the same application
  - All applications must see the same data
  - Data must be highly available
  - Must scale to support growth in the amount of data stored
  - Must scale to support growth in the number of clients accessing the data

- ObjectGrid function
  - Clients connect to ObjectGrid servers to access data
  - Multiple applications can access data transactionally
  - Data replicated between ObjectGrid servers
  - Data partitioned across ObjectGrid servers
As a Database “shock absorber”

- Application requirements
  - Database operations must be offloaded from the database
    - Improves performance
    - Reduces costs
- ObjectGrid function
  - Transactional access to the database
  - ObjectGrid servers receive data from the database and cache results
    - Look-aside or write-through cache can be implemented
  - ObjectGrid clients retrieve cached results
Database “shock absorber” Example

For 3-tier architectures, significantly speedup database access…

Reduced bottleneck of database transactions by up to 95% (based on IBM experience)

Multiple clients are relieved of direct access to the database, removing contention

Cache front-ends a heavily used database; information is committed from the cache to the database as required
As a Multi-platform cache / data source aggregator

- The ObjectGrid can provide a shared multiple quality of service cache between multiple platforms
- Objects stored in the ObjectGrid can be shared between the following platforms:
  - WebSphere V5.1.x and stack products
  - WebSphere V6.x and stack products
  - Geronimo
  - Tomcat
  - Non IBM commercial and open source Java application servers
  - Standalone Java applications
- ObjectGrid can be customized to aggregate data from multiple data sources
  - Requires custom code
As a Scalable, low latency, highly available data store

- Application requirements
  - Extreme scalability
  - Low latency data access
- ObjectGrid function
  - Provides capability to execute logic local to the data
  - Eliminates network latency
  - Eliminates serialization
  - Application lifecycle coupled to the placement of data
  - Requires use of the DataGrid APIs in the ObjectGrid application
WebSphere eXtreme Scale as a Cache

Frame

Router

A-I

J-R

S-Z

Server
Data Grid near-cache

Records A-I

CPU

Server
Data Grid near-cache

Records J-R

CPU

Server
Data Grid near-cache

Records S-Z

CPU

DG Server

Records A-M

CPU

DG Server

Records N-Z

CPU

Database
As a HTTP Session Persistence Mechanism

- HTTP Sessions can be replicated across servers using ObjectGrid
  - A Servlet filter that enables session replication can be inserted into any Web application
  - Provides a Session persistence approach that is independent of the WebSphere cell infrastructure

- High speed, local access to ObjectGrid
  - The session filter hosts session state in the local process which promotes low latency read operations and low latency write operations when running asynchronous replication.

- Older versions of WebSphere products can use ObjectGrid as an upgraded session persistence mechanism

- Non-WebSphere servers (such as Geronimo or JBoss) can also use this Servlet filter
WXS Architectural Essentials
WXS – Dynamic Distributed ObjectGrid

Build ObjectGrid application using various APIs

- ObjectMap APIs
- EntityManager APIs
- DataGrid APIs
- Query APIs

ObjectGrid CatalogServer

ObjectGrid

- ObjectGrid Container
  - Partition 1
  - Partition 2
  - ObjectGrid Plug-in

- ObjectGrid Container
  - Partition 1
  - Partition 2
  - ObjectGrid Plug-in

configure

objectGrid.xml
objectGridDeployment.xml
entity.xml

Develop custom ObjectGrid plug-ins when required, for example:
- Loader plug-in
- Evictor plug-in
- ObjectTransformer plug-in
ObjectGrid Programming Model

Application Server

Application

JVM Heap

ObjectGrid

Session

ObjectMap

BackingMap
ObjectGrid Basic Application Flow

Application

ObjectGrid

Session

ObjectMap

BackingMap

Basic Application Flow

• Create ObjectGridManager
• Create ObjectGrid
• Define 1 or more BackingMaps
• Initialize the ObjectGrid
• Create a Session
• Create one or more ObjectMaps
• Begin the Session
• Insert objects into the ObjectMap
• Get objects from the ObjectMap
• Commit the Session
• Remove (destroy) the ObjectGrid
WXS Basics

• An ObjectGrid is a logical grouping of data

• An ObjectGrid server
  – Is a JVM running on a particular host and port
  – Hosts ObjectGrid containers

• An ObjectGrid container holds data in a collection of maps
  – In the programming model, these are called BackingMaps
Maps, Map Sets and Schemas

- Maps are grouped into map sets
- Map sets have schemas
- Schemas define the relationships between the elements in different maps
Data in a Map Set can be grouped into Partitions

- Each piece of data exists in exactly one partition
- Each partition can be replicated to multiple servers
- Data related by schema are all in the same partition
- Note: Some applications may allow only one partition
  - This is not as scalable
Shards

- There is exactly one primary shard
  - There can be zero or more replica shards
Shard Types

• Primary (P)
  – Receives all insert, update and remove operations.
    • Replicas are read-only
  – Adds and removes replicas
  – Replicates data to the replicas
  – Manages commits and rollbacks of transactions
  – Replicas turn into primaries to handle failure events or rebalancing

• Synchronous replica (S)
  – Maintains the same state as the primary
  – Receives updates as part of the primary’s transaction to guarantee consistency

• Asynchronous replica (A)
  – Might or might not be at the same state as the primary
  – Receives updates after the transaction commits on the primary
    • Primary does not wait for the asynchronous replica to commit.
Zone Based Replication

- Zone support allows for rule-based shard placement, enabling high-availability
- Appealing to enterprise environments that need data replication and availability across geographically dispersed data centers
- Best-Practice: place SYNCH replicas in same zone and ASYNCH replicas in different zone for optimal replication performance and scaling across geographies and data centers
- *Zone support is a significant competitive differentiator in the in-memory data grid space*
Non invasive middleware

- **Single JAR, 13MB**
  - It has no WebSphere ND dependency and works with:
    - Current and older versions of WebSphere ND and CE
    - competitive application servers.
    - Straight J2SE
    - Spring
    - Sun(*) and IBM JVMs

- While ObjectGrid is self contained it requires an external framework for installing applications and start/stop the JVMs hosting those applications:
  - WebSphere XD
  - WebSphere ND
  - WebLogic, JBoss
  - Third party grid management software
The Anatomy of an ObjectGrid Application

So, what does an ObjectGrid application conceptually look like?
Basic Implementation Scenarios
Scenario 1: Side cache

- Here, the grid is used by the application as a coherent distributed cache.

- Every cache lookup is an RPC to the server that can hold that key.

- If the data isn’t there then the application gets the data normally and then stores it in the cache for next time.

- Applications can use this directly or use L2 cache plugins for popular object relational Mappers like OpenJPA or Hibernate.

Slower than a local HashMap BUT
- faster than the backend
- the cache can be huge
- all cache clients see the same data guaranteed
- Data is already in object form
- Offloads the backend
- No more cold caches on JVM start
ObjectGrid Basic WAS Web Application

1. Client A – get flowers
2. Servlet – creates the Session
3. Servlet – gets the ObjectMap
4. Servlet – begins the Session
5. Servlet – attempts to get flowers
6. ObjectMap - returns null (cache miss)
7. Servlet – retrieves flowers from DB
8. Servlet – inserts flowers into ObjectMap
9. Servlet – commits the Session
10. Servlet – returns flowers to Client A
11. Client B – get flowers
12. Servlet – creates Session

1. Servlet – gets the ObjectMap
2. Servlet – begins the Session
3. Servlet – attempts to get flowers
4. ObjectMap – returns flowers (hit)
5. Servlet – commits the Session
6. Servlet – returns flowers to Client B
First, what's a Loader

- A Loader can be provided by IBM or written by the customer.

- It provides a delegate to the backend for the WXS.

- If data cannot be found in memory then WXS asks the Loader to get the data if possible.

- All changes are provided to the Loader by WXS.

- Automatic Loaders are available for DB2 and other databases.
Scenario 2: Scenario 1 + Sync Loader

- This is the same as scenario 1 but the application associates a Loader with each Map in the cache.

- The applications looks up a key and if the key isn’t in the cache then the Loader is invoked to pull it from the backend. This is a more efficient mechanism than before, fewer RPCs (two versus three).

- Changes are written to the cache and the Loader is called synchronously to write the changes to the backend.
WXS application flow with Loader

1. map = session.getMap();
2. map.get(key1);
3. map.update(key1, value2);
4. session.commit();
Scenario 3: Scenario 2 with write behind

- Same as two but:
  - Usually all the data is preloaded into the grid
  - The grid becomes the system of record
  - Changes are written to the grid and replicated synchronously.
  - Periodically, the changes are flushed to the Loader and so to the backend.
  - If a record is updated multiple times during the period then only the latest version is written.
  - If the backend is down then it just tries again later.

- Writes scale linearly with this approach and the backend load is significantly reduced as there are fewer larger transactions.

- Backend availability has no impact on application availability.
Scenario 4: Collocated application

- Normally, applications are stateless and leverage the grid for all their state.

- This is faster using a database for the state but still requires an RPC for each data access.

- If the application logic runs within the same JVMs as the data AND the requests for data in a particular partition can be routed to the JVM holding the data then this hop can be eliminated.

- This approach results in best application performance, especially when combined with write behind.

- Our HTTP session manager uses this approach.
Data Grid Application Patterns

- The DataGrid APIs allow for running business logic close to the data
  - Large sets of data can be partitioned and spread across many JVMs
    - Distributed ObjectGrid
  - Running business logic **locally** on each partition
    - No serialisation/deserialisation overhead!
    - No additional network hop!
  - Business logic runs **in parallel** on each partition
    - Allows for linear scalability for a whole set of problems
    - Valuable for fast processing times

- The DataGrid APIs support two patterns
  - Parallel Map
    - Allows the entries for a set of Entities or Objects to be processed and returns a result for each entry processed
  - Parallel Reduction
    - Processes a subset of the entries and calculates a single result for the group of entries
Data Grid Agents

Agent is serialized to every grid container (or a subset)

Results
Single result or multiple results returned to client as desired
Customer Samples
Retail Bank – Database Cache

Problem
AA retail bank is currently using a database as a service cache for a backend application. A daemon removes old entries using a cron scheduler. The cache currently eliminates 50% of the load on the backend.

- Single point of failure for the application tier
- Performance bottleneck
- Expensive vertical scaling is expensive.
- Expensive solution for the storing data with a life time of minutes.

Solution
- Use WXS as the cache

Key Features / Benefits
- Inherently scalable through horizontal scaling.
- Low cost/low TCO platform compared with a database.
- Predictable scaling costs.
Retail Bank – Mainframe MIPs Reduction

Problem
- Applications used a common profile service to retrieve customer profiles
- Individual applications use private caches to store profiles causing a siloed cache environment and unnecessary profile fetches
- Customer wants to eliminate redundant profile fetches to better leverage their 390 investment

Solution
- WXS is used as a network attached Data Grid holding around 8Gb of profiles (4GB + 4GB for high availability).
- A mediation is inserted in the ESB to memorize profile fetch service calls.
- The service name and parameters are used as a key and the value is the profile itself.
- If the profile isn’t in the cache then the mediation hits the 390 and stores the result in the Data Grid.
- An evictor removes entries older than 30 minutes.

Key Features / Benefits
- Reduction of roundtrips to the mainframe
- Improved Performance
- More optimal use of 390 MIPS
- Ability to handle greater transaction volumes
Scalable Web Profile Service

Problem
- Web site has a global audience and is served by five data centers from the USA to Asia.
- The user profiles are kept in a database in the USA. Replicas are available in the other data centers but updates must be on the US system.
- There are around 10 million registered users and profiles are updated frequently. Profiles are around 10k per user. 100GB total data.
- Issues with the US system becoming unavailable lead to outages
- Multiple applications looking for Profiles

Solution
- Profiles loaded from the database at startup
- Cache is the state-of-record for the data
- Write Behind to database to batch changes
- Reload from database on catastrophic failure
- Zones used to determine placement of replicas
- This is a common scenario for very high activity web sites
- Solution used also for cache loaded from a stream of events

Key Features / Benefits
- Ability to handle increasing volumes of data
- Scalable latency for data
- Reduced database load with use of the write-behind feature
- Fault tolerance
Scalable HTTP Session replication between data centers
Scenario

- A large company has a web portal that runs in two data centers.
- They tried to use DRS for HTTP session replication but it’s problematic to setup on large clusters and can’t handle multiple data centers.
Solution

- Customer will deploy a cell per data center. The application is deployed to both cells.

- Apace HTTPD acts as a HTTP sprayer for each data center.

- ObjectGrid Session manager is plugged in to the web application using a servlet filter.

- The catalog service is deployed across both data centers and both cells share a single catalog service, binding both together as a single grid.

- Each cell is marked as an individual zone and replication rules are put in place to place the primaries and asynchronous replicas in different zones.
Multi-cell HTTP Session Topology

- Multiple cells share a single catalog server
- Deploy the application to each cell
- Zone rules can be used to influence placement for the desired availability
- Shared catalog can be:
  - Its own cell
  - Running in J2SE JVMs
Http Session Replication and Failover

**Problem**
- Web Applications need failover capability for sessions across datacenters

**Solution**
- Use WXS in conjunction with Zone Rules feature.
- Used in Retail Banks
The Caching Landscape
Market Dynamics

- **GigaSpaces**
  - Enterprise Application Grid
  - Revenue ~ $20M

- **GemStone**
  - GemFire Enterprise Data Fabric, GemFire Real Time Events
  - Revenue ~ $10M

- **eXtreme Scale**

- **ORACLE**
  - Coherence
  - Revenue ~ $30M - $40M
### Oracle Coherence: “800-lb gorilla”

**Strengths**
- Awareness and mindshare - #1
- Natural avenue to attack IBM’s transaction processing leadership
- Development roadmap to integrate across Fusion middleware (“SOA Data Grid”)

**Weaknesses**
- No asynchronous replication
- No data placement by locality (zones)
- Positioning with RAC and Oracle DB franchise can be troublesome

### GigaSpaces

**Strengths**
- Mindshare in XTP space, gaining traction
- Partnership with Spring

**Weaknesses**
- Immaturity and small company characteristics weigh against enterprise adoption
- Management console with SLA Mgmt

### Terracotta

**Strengths**
- OpenSource Source with an option to buy support

**Weaknesses**
- Being Open Source weighs against enterprise adoption
- Central Server model, where server becomes single point of failure and reduces scalability
- Not scalable for large datasets
- No Application Programming Interface

### GemStone

**Strengths**
- Well established in marketplace with over 200 customers
- Mindshare in Financial Services space

**Weaknesses**
- Small company characteristics weigh against enterprise adoption
- No Entity Manager API
In-Memory Database - SolidDB

- IBM SolidDB is a relational in-memory database
- Applications can take advantage of its capability through standard ODBC, JDBC and SQL interfaces
- Extremely fast as data resides in memory at all times rather than on disk
- Is highly available because it maintains two copies of data synchronized at all times
- Can partition the tables into multiple instances scaling-out across multiple servers
SolidDB Vs WebSphere eXtreme Scale Positioning

- For adding simple caching to a new and existing system both WXS and SolidDB can be used.
- For systems that need large scale-out, eXtreme Scale is the better fit; it can linearly scale transaction processing systems out to thousands of servers.
- For SQL-Intensive applications with complex SQL query requirements, but no extensive scale-out SolidDB is a good fit.
- For systems needing to cache results from backend services, eXtreme Scale is a better fit. Its Loader interface permits caching results from services, accelerating performance and adding scalability to a SOA.
Main competitive benefits

- Embeddable and it doesn’t require a new ‘platform’.

- Also tightly integrated with all WebSphere ND versions from V6.0.

- State of the art programming model using EntityMananger for transparent POJO persistence.

- Working with Object graphs is directly supported using the EntityManager APIs.

- State of the art replication technology including industries only multi-data center capability.

- Uses just TCP for communication, no multicast or UDP.

- *Low TCO*
WXS Application Architecture Summary

The resulting architecture is:

• **Linearly scalable** -- because there is no dependency among the partitions, the law of diminishing returns does not apply. Each additional unit added provides the same throughput as the one before it.

• **Low latency** -- because network hops between the tiers and the services that make up the application have been eliminated, and because data and events are accessed locally and in memory.

• **Simple** -- because you have a single clustering model to manage high-availability, load-balancing and partitioning across your entire environment.
Resources

- Melissa Pike’s Team and the WAS Infrastructure WIKI
  http://w3.webahead.ibm.com/w3ki2/display/WAIInfTS/Home

- Fully functional J2SE trial download

- Wiki based documentation
  http://www.ibm.com/developerworks/wikis/display/objectgrid/Getting+started

- Billy Newport’s Blog  http://www.devwebsphere.com

- WSTC Presentations
Thank You – Questions?
Generic Application Architecture
Quantifying Caching Benefits
Questions to ask customers?

- Is the application making too many expensive roundtrips to the backend database?
  - Are the database accesses redundant?
  - Is the same data being accessed by multiple applications?

- Is the data accessed mostly read only?

- As the data volume grows exponentially, how will the current infrastructure scale?
  - Is the growth affecting your data access time and/or throughput

- Are you spending a large number of resources in managing the environment?

- Is the volume of data so large that it cannot possibly be stored in a single physical box? Or in a traditional database?

- Is there a requirement for extreme levels of data processing and real time analysis of gigabytes of data with low latency, constant millisecond level response time?

- Are you comfortable with your high availability strategy?

WebSphere eXtreme Scale can help
Data Access time (ms) = 
(Probability of near-cache hit) * (Time to retrieve data from near-cache) + 
(Probability of near-cache miss) * (time to retrieve data from other storage);

Time to retrieve data from other storage (ms) = 
(Probability that data is in cache server) * (Time to retrieve data from cache server) + 
(Probability that data must be retrieved from database) * (time to retrieve data from database);
Example calculation

Data Access = (Near-Cache Hit) + (Near-Cache Miss)

Near-Cache Hit = (P1) * S1
Near-Cache Miss = (P2) * (P3) * S3 + (P4) * S4

Near-cache miss = .7 * 10 + .3 * 200
= 7 + 60 = 67 ms

Data Access = .3 * 1 + .7 * 67
= .3 + 46.9 = 47.2 ms
Example calculation - effects of increasing the cache size

Data Access = (Near-Cache Hit) + (Near-Cache Miss)

Near-Cache Hit = (P₁)(S₁)
Near-Cache Miss = (P₂) * [(P₃)(S₃) + (P₄)(S₄)]

Near-cache miss = (.7)(10) + (.3)(200)
               = 7 + 60 = 67 ms

Data Access = (.6)(1) + (.4)(67)
             = .6 + 26.8 = 27.4 ms

(47.2 - 27.4) / 47.2 = 42% improvement in data access time
Thank You

We appreciate your feedback in order to improve this educational event. Please fill out the survey form.