ORACLE

Architecting Applications for Scalability, Performance and Availability

Oracle Coherence Workshop

ORACLE Coherence

Agenda

- Understanding "Scalable Performance"
  - Scalability Refresher
  - Scalability Approaches
- Scaling the Application Tier
  - Scaling Without Coherence
  - Scaling With Coherence
  - Why Is Scaling Out the Application Tier Hard?
  - Is Clustering Always the Answer?
- Coherence's Approach to Scalability
  - Coherence Cache Topologies
- Q&A

Understanding Scalable Performance

Scalability Refresher

Performance is Like a Ferrari

- Performance is like a Fast Car
  - Designed for speed (not capacity)
  - Improve engine and components (**scale up**)

Scalability is Like a Train

- Scalability is like a Locomotive
  - Designed to handle load and capacity
  - Add more cars and engines (**scale out**)

Scalability and Performance don't always come together

- You can't just add them together!
  They have to be designed.

Scalability and Performance – Sometimes They Do Come Together

- Again – They have to be designed.

Application Scalability

- Scaling the Application-Tier is difficult
- If it was easy it would be an IDE option

Not possible!

- Scalability is a design option
  - Requires knowledge, care and experience
  - Developers have the "option" to consider building it in!
  - It's not an IDE option
- Coherence is scalability infrastructure for the application-tier

What is Scalability?

- Scalability:
  "The degree to which the performance of a system improves when more resources are added"
- Linear Scalability:
  "When resources are increased by a factor of n, system performance improves by the factor of n"
- Predictable Scalability:
  "The ability to know in advance of adding resources the degree to which a system will scale"

Scalability Approaches

Approach	How	Advantages	Disadvantages
Vertical "scaling-up"	Increase resources in existing server(s)	❖ Relatively simple process (can be achieved overnight) ❖ Transparent to system architecture and development	❖ Comparatively expensive hardware (niche) ❖ Limited Scalability (physical limits typically encountered) ❖Increases cost of failure
Horizontal "scaling-out"	Add more servers	❖ Comparatively inexpensive hardware (commodity) ❖ Virtually unlimited scalability possible (typically greater than scale-up approach)	❖ Applicable only when a system is designed to "scale-out" ❖ May require months of rework to achieve ❖ Scalability may be limited by "network" ❖ Requires additional administration

Developers and Scalability

- Be aware!
  - Poorly designed algorithms and data structures may not scale
  - Scalability is often a non-functional requirement
  - Scalability is often "left to last" and not "designed in up-front"
  - Developers tend to assume that their system is scalable
  - Developers are often surprised that their system is not scaling
  - Developers tend to assume there is a quick fix for scaling
  - Developers may assume Coherence is a drop in solution
  - Coherence may not be a solution (often it is… more later)
  - While a system may be scalable, often operational costs are not taken into account (it's someone else's problem)

What do we mean by "Scalable"?

- High scale
  - Scales readily to ~100 servers
  - Practical limit of ~1000 servers
  - Support for thousands of simultaneous clients
  - Multiple Sites
  - Across continents & globe
- Easy scale
  - Just plug in additional machines
  - While system is running
  - No need for manual application partitioning

What do we mean by "Performance"?

- Instant access
  - Clients can maintain coherent data in local memory
  - Faster than disk or even network
- Instant awareness
  - Clients can subscribe to real time events
  - Notification to application servers or even desktops
- Parallel data processing
  - Clients can push processing to the servers
  - No data movement results in very high performance

Defining "Scalable Performance"

- ***Performance*** **describes the elapsed ("wall clock") time that it takes to execute an operation**
- ***Scalability*** **describes how a system behaves given an increasing number of simultaneous operations**
  - In scalability terms, predictability is more important than the raw performance exhibited for any single operation
- ***Scalable Performance*** **describes a system that can scale *predictably* under load, and can also execute operations quickly**
  - Provides predictability to the cost of scaling up an application

Scaling the Application Tier

(Without Coherence)

Scaling the Application-tier (without Coherence)

Approach	How	Advantages	Disadvantages
Scale-Up *"It's an infrastructure problem"*	❖ Buy Big Boxes ❖ Increase Resources (cpu, memory, hdd capacity, speed and network, etc) ❖ By specialized hardware (Azul, Infiniband…)	❖ Simple (overnight) ❖ No development ❖ No impact on internal design	❖ Expensive ❖ Will hit physical limits ❖ Will have to redesign at limit ❖ Non-graceful deterioration at limit ❖ Stop, Add, Restart required to scale

Scaling the Application-tier (without Coherence)

Approach	How	Advantages	Disadvantages
Stateless Scale-Out *"Push state scale-out into lower Data Source layer"* *"It's the DBA's problem"*	❖ Make application stateless (eg: stateless sessions) ❖ Use lots of stateless servers ❖ Use load-balancing ❖ Use "big" and "scalable" Data Source to ensure application state scale-out	❖ Easy to develop (not overnight, but relatively simple as no state is managed) ❖ Scale-out is easy, just add more servers	❖ Only scales to match underlying Data Source performance ❖ When underlying limit is reached, have to redesign ❖ Network bottlenecks experienced as data is moved between layers

Scaling the Application-tier (without Coherence)

Approach	How	Advantages	Disadvantages
Caching *"Keep recent copies of state"* *"We'll save the DB and DBA by caching"*	❖ Application keeps local copies (in memory or on local disk) of recently / commonly used state	❖ Seems simple ❖ Reduces Data Source and Network load ❖ Significant application performance improvements	❖ Maintaining consistency of data between Local and Data Source instances can be difficult ❖ Require "messaging infrastructure" to ensure coherency across a cluster (and application development) ❖ Typically applicable to "read only" applications and not "write a lot" applications ❖ Easy to get wrong

Scaling the Application-tier (without Coherence)

Approach	How	Advantages	Disadvantages
Use an Application Container *"Our magical clustered container will scale our application infinitely"*	❖ Believe the vendors & the marketing ❖ Follow a "scalability paradigm" ❖ Use a "Clustering Container" … It scaled the "Pet Store" linearly, therefore our X application will also scale linearly (where X ≠ "Pet Store)	❖ Simple ❖ Well documented and communicable paradigm ❖ Easily scale development team	❖ Typically scales in-the-small ❖ Usually relies on "scale-up" rather than "scale-out" ❖ Requires specialized skills or products (out side of the standard paradigm) to really scale ❖ Clustering is primarily about High-Availability, not Scalability!

Scaling the Application-tier (without Coherence)

Approach	How	Advantages	Disadvantages
Manually partition the Application and / or Data *"Scalability is easier in small bits"*	❖ Break the application domain into independently scalable components ❖ Have separate teams deal with their own components ❖ Use "pools" of Services to perform work ❖ Use load-balancing to scale-out	❖ Seems simple ❖ The problem isn't as big as it was before ❖ Some components may actually scale better by themselves	❖ Often difficult to decompose the application ❖ What's good for one component, is often bad for another (eg: if you need 'joins') ❖ Typically introduces new bottlenecks (sharing information between components) ❖ Managing an application composed of many independent parts is more complex!

Scaling the Application-tier (without Coherence)

- In summary…

  "Solving application-tier scalability is either;
  a). someone else's problem, or

  b). involves the complex process of partitioning and managing data, services and coherency across a collection of servers."

- Coherence provides developer solutions for b) to enable predictable application scale-out

Scaling the Application Tier

(With Coherence)

Scaling the Application-tier with Coherence

Approach	How	Advantages	Disadvantages
Use Coherence to share and manage objects (application state) *"Coherence is responsible for my objects"*	❖ Introduce Coherence libraries into Application(s) ❖ Use Coherence NamedCache API (derived from java.util.Map) to store application state ❖ Start multiple Coherence-enabled processes to scale-out (load balance) objects (data)	❖ Simple ❖ Transparent and Automatic Partitioning of Data ❖ RemoteException-free distributed computing ❖ Itself is massively scalable ❖ Displaces other technology (messaging) ❖ Extremely configurable	❖ New paradigm ❖ People tend to use old patterns with it – that don't work or are overly complicated ❖ Configuration isn't easy (at first) mainly because of the new paradigm ❖ Takes time for people to "trust" the technology ❖ Extremely configurable

Scaling the Application Tier

Why Is Scaling-out the Application-Tier Hard

"Scalable Performance" is hard…

- **Scalable performance is the black art of making something go fast while also being able to handle more load in a *predictable* fashion.**
- **Often, scalability features will conflict with raw performance features**
  - Bite the bullet up front: Architect scalability from the get-go
- **It is tough to try to decide (up front) between scalability and raw performance**
  - If scalability could be necessary, it needs to be architected in
  - Most applications don't actually need to scale, though!

Why Scaling-out the Application-Tier is Hard!

- However…

  "It's extremely difficult to write software that ensures an unpredictably (dynamically) growing collection of servers connected by an unreliable network can continuously work together without losing information (or work) in a manner that itself is linearly scalable"

- Significance…
  - Achieving all of these things in the same product
  - Working together means "consensus" has to be maintained!

Obstacles to Scale

- **Resource Usage: Later Tiers**
  - There is a cost when a tier invokes a later tier
  - Collocation of tiers reduces inter-tier communication
  - Applications that have to talk to the database on each request will suffer from scalability problems
  - The Database tier is *difficult* and *expensive* to scale; it is difficult to scale a database server to more than a single host, and it becomes exponentially more expensive to add CPUs
  - Database servers scale sub-linearly *at best* with additional CPUs, and there is a CPU limit

Obstacles to Scale (Summary)

- Architect so that the application is CPU-or memory-bound, and that the bottleneck is in the application tier at the latest
- For high-scale applications, make sure that the bottleneck will never be the data source (mainframe service, database)
- Benefit: You can use server farms and server clusters to scale an application almost linearly and with a predictable cost per user

Is Clustering Always the Answer?

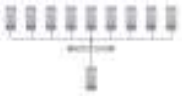
- Clustering enables multiple servers or server processes to work together
- Clustering can be used to horizontally scale a tier, i.e. scale by adding servers
- Clustering usually costs much less than buying a bigger server (vertical scaling)
- *Clustering also typically provide failover and other reliability benefits*

Clustering Categories

- **Master/Slave:** For availability
- **Parallel:** For scalability, e.g. stateless web server farms
- **Centralized:** Single server for coordination *(can represent bottleneck and/or SPOF)*
- **Hierarchical:** Multi-tiered centralized model
- **Peer-to-Peer:** Servers work independently, but have knowledge of and direct access to the entire cluster *(cooperative worker model)*

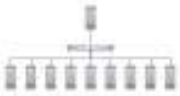
Traditional Scale-Out Approaches…

#1. Avoid the challenge of maintaining consensus
- Opt for the "single point of knowledge"



Client + Server Model (Hub + Spoke)

Active + Passive (High Availability)



Master + Worker Model (Grid Agents)

#2. Have crude consensus mechanisms, that typically fail and result in data integrity issues (including loss)

Traditional Scale-Out Approaches…

Real-World Feedback
- Have unbalanced / unfair load and task management
  - Some servers have greater system responsibility than others
- Have Single Points of Bottleneck (SPoB)
- Have Single Points of Failure (SPoF)
  - "Micro outages" are magnified as you scale-out
- Exhibit Strong Coupling to Physical Resources
  - Software completely dependent on individual physical servers
- Require specialized deployment and operation for individual Resources
  - Some servers require "special attention" to operate
- Traditional scale-out approaches limit
  - Scalability, Availability, Reliability and Performance

Best Scale Out Approach – What Do We Really Need?

- **Scale Out**
  - Something that Performs Better when you scale out
  - Something that does us makes us **not** choose between stateless and stateful approaches
- **Clustering**
  - Something that takes advantage of peer to peer clustering (no master/slave, of hub)
- **Clustered Caching**
  - Something that also clusters objects across nodes (cache clustering)
- **Finally – Scalable Performance:**
  - **We Need Clustering, Scaling and Caching to be all Possible at the Same Time**
  - **We Need all 3: Reliability, Performance and Scalability, without sacrificing any one.**

Best Scale Out Approach

- Discussion on how Coherence Does This is Next….
  - Clustering, Scaling and Caching to be all Possible at the Same Time
  - Reliability, Performance and Scalability, without sacrificing any one.

Copyright 2007

BREAK

Copyright 2007

Q&A

Copyright 2007

ORACLE