How Cloud is working as a disruptor to shake up middleware design

EVOLVE OR DIE!

Billy Newport (@billynewport)

IBM Distinguished Engineer

Creator of IBM WebSphere eXtreme Scale

Agenda

- **Talk about the environments for traditional middleware**
- Discuss design of traditional middleware
- Discuss the cloud based environments
- Discuss the impacts on making middleware fully elastic for the emerging cloud based environments.
- Questions

Traditional Environments

- Projects purchase dedicated boxes for their needs only.
- These boxes are assigned permanent network addresses, host names and installed in a data center.
- They stay there until they break or upgraded.
- These boxes are thought of as permanent.
- The disks can be used for long term storage.

Traditional Middleware design

- These assumptions had consequences on middleware design.
- Middleware was designed to know information about each box, host names, ip addresses.
- Each box can store state like message stores/transaction logs, diagnostics logs and so on.
- Customers as a result focused on topology as the main issue when deploying an application.
- Set up a cluster, add nodes, deploy app to cluster and so on.
  - Server images are HETEROGENOUS.

- Customer knows how to integrate pieces, database/messaging/ http, routing and so on. It's not automatic or cheap.

High availability

- Given boxes are permanent, many middleware products did not include high availability out of the box.
- Third party products (Sun Cluster, IBM HACMP/Tivoli TSA, Veritas, …) needed to be purchased, configured and this was a one time cost as once done, the boxes were used for years sometimes.
- This software wasn't cheap and many times required dual ported or SAN style disks to even operate.
- Rare events are not optimized for. Setup for example.

Removing Boxes or failed boxes

- Consequently, if you want to remove a box then:
  - You need to remove it from the topology for any middleware using it.
  - Take care any state remaining on it isn't useful any more.
  - You may need to reconfigure another box with the state from the failed box for recovery reasons (transaction logs, database/messaging state).
  - You may need to reconfigure high availability 'stuff'
- You cannot just turn a box off and forget about it.

Multiple data centers are rare

- Most middleware stacks are not designed to run active/active in multiple data centers.
- Data centers are expensive, only a few large customers have more than one.
- Typical design is:
  - a database in one data center
  - the application in the same data center.
  - Second data center is cold, maybe a replicating dbms is hot, that's it.
  - Second data center is usually used as test system.
- **Most vendors don't charge for software in a standby data center**.

Lots of memory available

- Customers bought boxes with all the memory they required for the application.
- Applications could be designed to use a lot of memory because they bought their own boxes.
- Applications typically use large intra JVM caches.
- People look to 64 bit to handle bigger caches.
  - 64 bit JVMs may have a short life once virtualization takes hold as we'll see later…

State = Database (no!)

- **When all you have is a hammer, everything starts to look like a NAIL!**
- Most applications use the database for all persistent state.
- This causes huge problems with multiple data centers because of network latency.
- People don't realize that you can use different types of storage depending on the persistence interval required.
- DataGrids offer elastic state solutions for data that needs to be fault tolerant and be shared between application(s) but doesn't need to be durable for decades!
- You can stage data in a DataGrid while you work on it and push it back when you're done.
- Not all state needs to be on a SAN.

No need to Partition Data Model

- There is only one database instance, right?
  - Design data models where everything refers to everything.
  - Transactions can touch all data.
  - No constrained tree schema.
- This will not scale out and is expensive to scale up.

Agenda

- Talk about the environments for traditional middleware
- Discuss design of traditional middleware
- **Discuss the "cloud based" environments**
- Discuss the impacts on making middleware fully elastic for the emerging cloud based environments.
- Questions

Disclaimer

- CurrVM:
  - I will be using CurrVM to indicate a consolidation based virtualization solution LIKE VMWare. It represents the current style of private virtualization.
- NextVM:
  - I will be using NextVM to indicate elastic virtualization solutions like EC2. It represents how I think private virtualization systems will evolve over the next 3 years.

On demand machines

- Machines can now be “created” and “destroyed” in seconds.
- Ideally, customers want to create machines when they are required and destroy them when they are no longer needed and do that as fast as possible.
- Traditionally, customers bought peak sized configurations resulting in low utilization.
- Now, customers only want to pay for resources for the application when there is work for those resources.

On demand data centers

- Data centers can be “created” in seconds.
- It’s a command line parameter when creating the virtual machine:
  - west coast or east coast
  - US or Europe?
- **This will mean more customers can use multiple data centers and as a result there will be more pressure on middleware vendors to support it than before.**

Virtual machine characteristics

- CurrVM:
  - Permanent. Each machine has its own disks even when its shutdown.
  - Server images are hetergenous.
  - Assigned a network IP address using DHCP or static but host name is usually well known.
- NextVM:
  - Temporary.
  - Each machine starts from an image and has disk assigned ONLY while it's running.
  - Server images are homogenous.
  - IP address is dynamic and assigned only while running.

Chargeables

- A virtual machine has the following resources:
  - Memory/network and processors
  - Starting disk image
  - Working disk image
  - Finished disk image
  - Network address
- People will get charged for ALL these resources.
- Therefore, the customer will want to be charged nothing or as little as possible when they don't need the box.

Storage types

- **Startup Image**
  Starting images can be stored on low performance, high capacity media (think S3).
- **Running Image**
  These can be copied to high performance disks for when the machine starts. (think SAN)
- **Stopped Image**
  Shutdown images can be stored on low performance, high capacity media (think S3).

- Low perf/high capacity = cheap
- High perf = expensive

Virtual Machine Chargeables

CurrVM

- Start/Running/Stopped disk image is the same.
- Customers will be charged for this storage until the virtual machine is deleted.
- Customers will be charged for network/CPU/memory while it's running.
- All disk is high performance.

NextVM

- Start (s3) and "running disk" (disk) are different.
- There is no "not running disk", running disk is destroyed when virtual machine stops.
- Elastic IPs are chargeable.
- Customers will be charged for network/CPU/memory while its running.
- Only "running disk" is high performance.

Traditional Middleware on CurrVM

- Very conventional environment.
- Create virtual machine, assign IP/host name. Install software, add to ‘cluster’.
- Use like a conventional machine.
- Virtual machines are usually long running.
- It’s about consolidation, not elastic scaling.
- Pay for running resources + not running resources.
- Basically, FRIENDLY to traditional middleware.

Traditional Middleware on CurrVM

- Shutdown a virtual machine but it still 'exists' so normal middleware still works. Restart machine image when needed.
- Disk space needed for EACH virtual machine instance as server images are HETERGENOUS!
- Normally, all disk space is on high speed disks.
- Pretty traditional in fact, main issue is memory usage.
  - You can consolidate CPU/network
  - You can't consolidate memory or swap at your peril!
- Memory is an issue, work is needed to get the memory footprint lower to make the numbers work (see blog @ http://digg.com/u1Ab6b)

Traditional Middleware on NextVM

- Very different.
- Server images are homogenous.
  - Software needs to be installed in an AMI that can be reused for many virtual machine instances. It's not specific to one cluster or deployment.
  - Template model versus instance model. When a virtual machine stops, it's GONE, host names/disk/ip address all invalid/deleted.
- Middleware topology information gets confused easily.
- We need more of a service based view with location transparency.

Evolved Middleware on NextVM

- One image PER middleware version level NOT per cluster member instance.
- Customer pays low cost for start image (S3)
- Customer pays running costs.
- Customer pays nothing when machine is not running.
- Data Center support essential because the interested customer audience is MUCH larger now.
- These characteristics (especially the temporary nature) means it's a bigger challenge than CurrVM or conventional private data center.
- Memory remains an issue, work is needed to get the memory footprint lower to make the cost numbers work.

Comparison on cost

- NextVM style looks cheaper.
  - One image per middleware version versus one image per ‘permanent’ machine.
  - Images are ‘instance stateless’ in the sense of how it gets used. VMWare images are customized and permanent to their usage.
  - You pay a small constant storage charge for the middleware AMI and then only for resources that are being used. If it’s stopped, you pay very little.

- But, only if middleware evolves to play well in this type of grid.
- You need different or evolved middleware to get a low TCO here.

Traditional Middleware on NextVM Problems (sorry, opportunities!)

- Traditional middleware doesn't like the NextVM model.
- It expects machines to be permanent.
- Data centers are supposed to be a niche and expensive.
- Machines vanishing forever on a regular basis is a problem.
- It was supposed to be OK to keep state on a box!
- Traditional middleware wants a virtual machine image (AMI) per server not per product/version pair.
- For both cases, memory footprint needs to be reduced.
  - Applications/Middleware cannot assume unlimited memory any more
- Therefore, if a NextVM model is what virtualization is headed for (and lets assume the lowest cost solution is the path here) then traditional middleware has higher TCO than if its deployed on physical machines.

- Middleware needs to <u>evolve</u> to run well on NextVM …

Elastic Middleware to the rescue

- We're on the verge of middleware starting to evolve to a more elastic form.
- Elasticity means the middleware can discover resources and manage them automatically without the user being involved in the management of specific machines or resources.
- Elastic middleware will have a cheaper TCO in fully virtualized elastic topologies because it was designed or redesigned to work with the new assumptions.
  - Incumbent middleware stacks will stack to have 'shades' of elasticity on their way to elastic nirvana.
  - New stacks will have a time to market advantage.
  - By the time customers are ready, either is fine.

Properties of elastic middleware

- Dynamically discovered virtual machines and incorporated in to the fabric automatically.
- Homogenous virtual machines: Self configuring from a standard startup image.
- Little or no need for resources when shutdown.
- Customer/admins don't worry about starting or stopping an element, it should be like magic!
- Application centric. Customers deploy applications and the middleware working with the provisioner assigns resources to run that application as load dictates.
- State needs to be somewhere else, a virtual machine disk is now like a RAM DISK!
- Application Centric, Topology free elastic middleware.

EXAMPLE OF ELASTIC MIDDLEWARE
IBM WEBSPHERE EXTREME SCALE

IBM WebSphere eXtreme Scale

- IBM has developed a mature DataGrid product called IBM WebSphere eXtreme Scale.
- This is a distributed, coherent, transactional, hash partitioned, replicated, elastic data grid.
- It supports clients for:
  - HTTP Session Replication
  - ORM L2 Caches
  - Application APIs
  - .Net Client
- IBM is actively integrating in to the our portfolio to add elastic state management to our solutions.

WebSphere eXtreme Scale

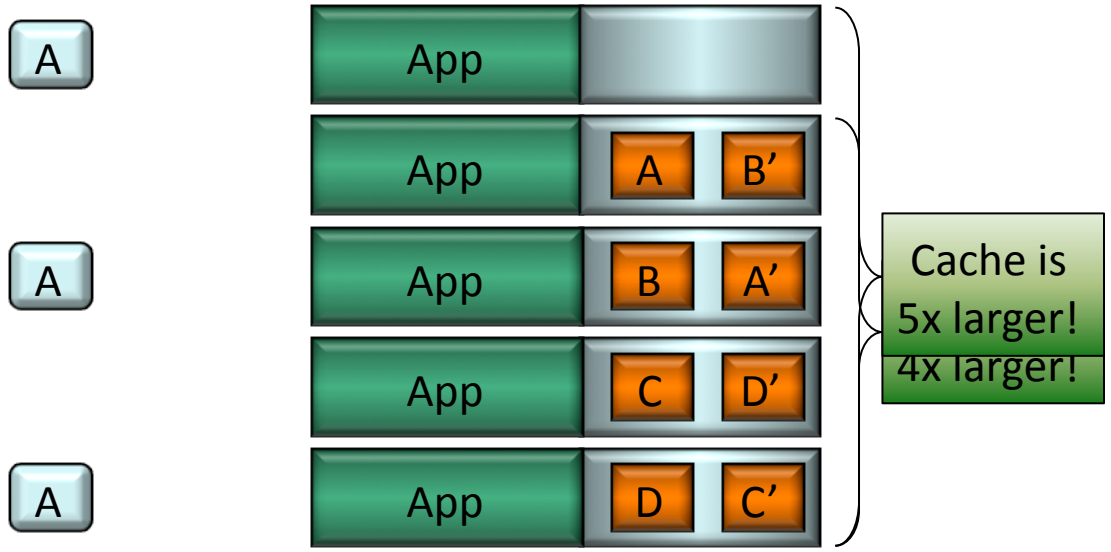
- The web application server can be modified to:
  - Use the WXS 'cloud' to manage HTTP Session state
  - Data caching
  - Other state normally kept on disk.

- The WXS 'grid' is fully elastic out of the box.

- WXS helps make the application server more elastic by allowing:
  - The application server to become largely stateless
  - By elastically managing state previously managed by the application server
  - By providing more appropriate storage choices for the application.
  - Scalable membership services which work across data centers.

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

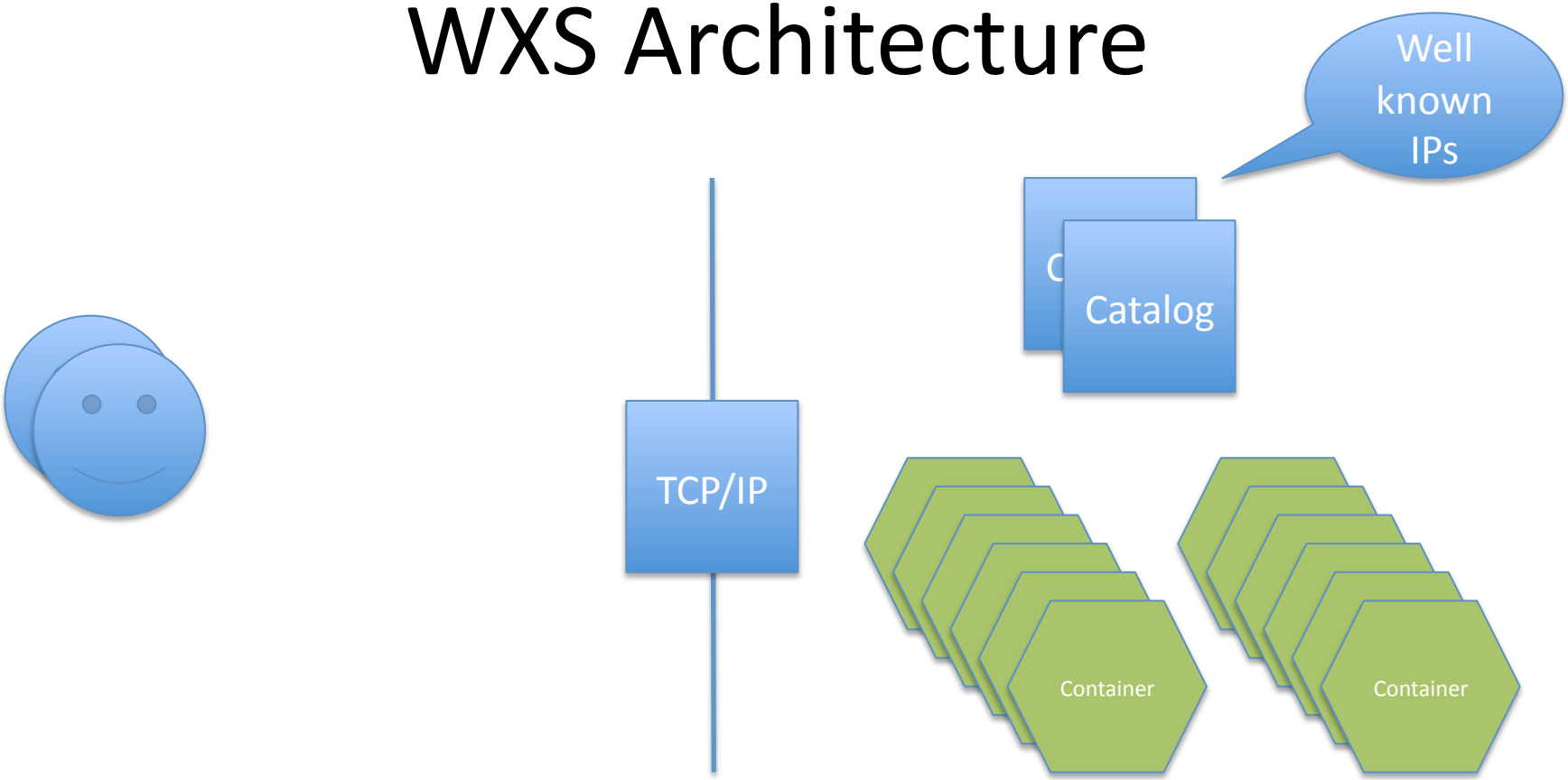


Topology free

- Customers don't have to manually add JVMs statically to groups and so on.
- Instead they define zone rules and a JVM is tested against these rules to dynamically group it from a topology point of view.
- Dynamic zone membership.

- Zones might be:
  - Blade Chassis
  - Data Center
  - Buildings
  - Floors
  - Physical machines running virtual machines

WXS Architecture



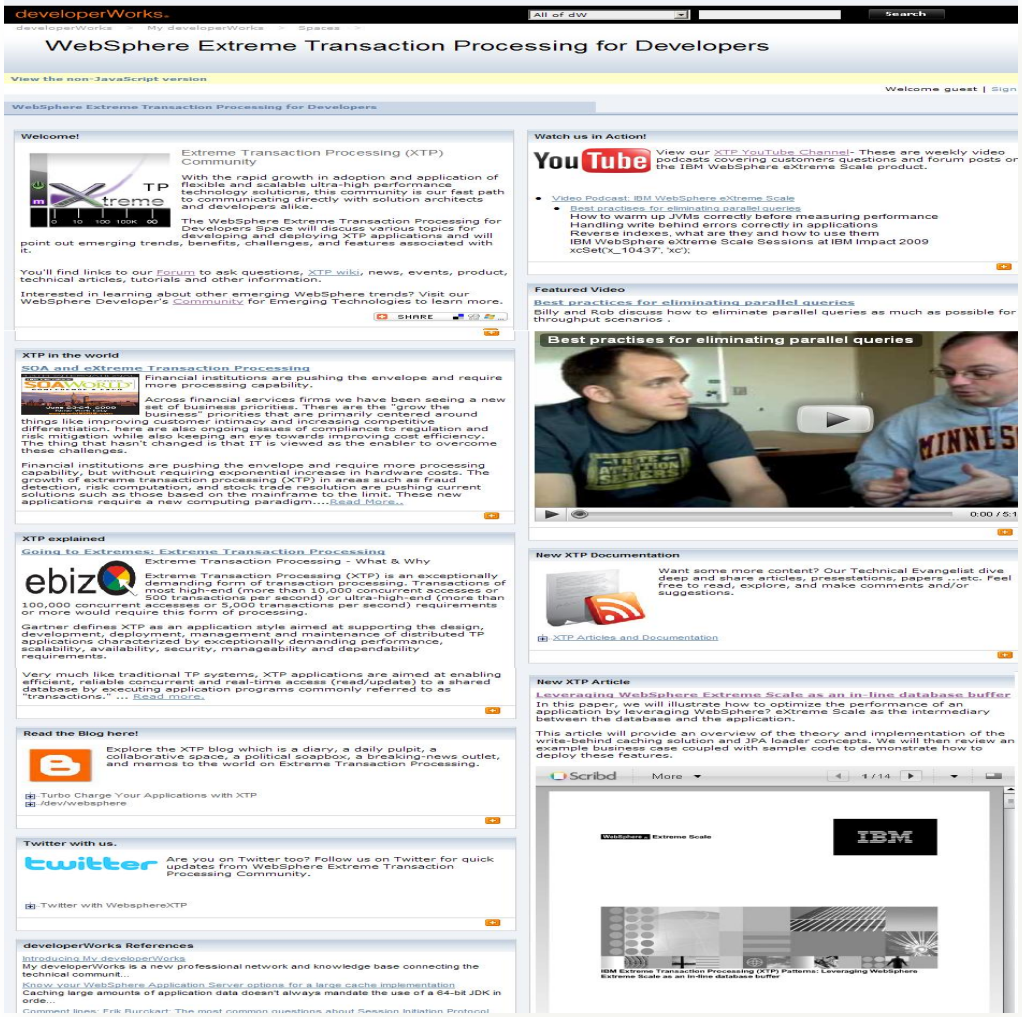
Only catalog IP addresses are ‘permanent’ and are all that’s needed to be well known. Everything else is dynamic. Only TCP used, no UDP/Multicast. One hop for all client operations. Geography aware routing.

IBM WebSphere eXtreme Scale IS ELASTIC

- It's designed to work in the future virtualization environments now.
- It's inherently highly available out of the box.
- It uses a fixed amount of well known network addresses no matter how large the grid becomes.
- It includes multiple data center support.
- Applications need to start leveraging DataGrids so they both perform better as well as work in advanced topologies.
- It's designed to work in a fully virtualized, temporary VM environment right now (homogenous server images).
- It manages the JVMs internally and automatically.
- Much of the current middleware state can be stored in this type of storage service.
- It can significantly reduce memory required (70%) when used instead of conventional caches.

More Resources – WebSphere XTP Community

developerWorks. **http://www.ibm.com/developerworks/spaces/xtp**



PROVIDES DEVELOPERS WITH

- Direct access to our technical evangelists and SMEs with channels such as:
  - Blogs
  - Twitter
  - Forum
  - YouTube Channel
- Prescriptive guidance for top scenarios.

Find latest news and collateral such as articles, sample code, tutorials and demos.

Questions and thanks

@billynewport