



The Cost Of New Technology

Simon Roberts

simon@dancingcloudservices.com





Libraries *et cetera*

- Libraries, frameworks, toolkits, whatever you call them
 - Intended to simplify programmer's lives
 - Are designed to be simple to use
 - Generally follow good OO principles
 - Increasingly, perform complex operations
- Hide the complexities of their behavior
- Discourage programmers from thinking about what the library does and how



Example Frameworks

- These exemplify principles of more general relevance
- EJB
 - Manageability
 - Security
 - Transaction framework
 - Memory management (object pools)
 - ORM
 - Legacy integration



Quality Of Service Issues

- Performance
 - Latency
 - Throughput
- Scalability
- Reliability
 - “Transactional” Correctness
- Availability



Quality Of Service Drivers

- Amdahl's law
 - Transactions
 - Synchronized blocks
 - Other constraints

- Persistence
 - Search efficiency/flexibility
 - Data integrity
 - Speed



Quality Of Service Drivers

- Network/Disk/DB I/O
 - Latency
 - Bandwidth
 - Concurrency
- Memory usage
 - Physical memory constraints
 - Garbage collector impact



EJB 2.x Transactions

- EJB 2.x entity beans “simplify” transactional correctness
 - Default behavior — everything is in a transaction, so developer “need not worry”
 - Creates illusion of 100% OO system — persistence “just happens”



EJB 2.x Transactions

■ Drawbacks:

- Unnecessary transactions — when entity is “live” DB must be locked
 - Even if you're only reading the data
- Transaction model is mapped directly to method model
 - Fine control is lost
 - Transactions last longer than needed
 - Transaction scope might be inadequate if “session facade” is not used
- (Might map methods to transaction model)



EJB 2.x Transactions

- More drawbacks:
 - Transactions are always propagated over a network
 - Network latency further increases duration of transaction
 - Access code is automatically generated, runs client side
 - Can't (generally) use stored procedures in CMP



Improving On EJB 2.x

- Newer frameworks return transaction control to the programmer
 - Along with the responsibility
- Most still make it hard to avoid client side transactions
 - And the network trips associated with this



Web Services & SOA

- Started out essentially without transaction support
 - Now being added in
 - But XML is a splendidly slow mechanism
- How smart is a transactional service?



Synchronized Blocks

- Early collection classes attempted to provide thread-safety
 - Liberally synchronized
- This protects the data structure, but not the higher level meaning
 - So you have to add more synchronization
- Newer collections don't synchronize
 - Concurrency is hard and usually, you just have to face up to this



ORM Systems

- Option 1: Use objects to create a schema?
 - Objects are designed without regard to use, schema usually requires denormalization for performance
 - All ORM tends to force client-side queries



ORM Systems

- Option 2: Connect objects to existing schema?
 - A denormalized schema is typically poorly supported
 - Much of the benefit of ORM may be lost if you have to patch up duplicate fields and indexes manually
- ORM has been described as the “Vietnam of modern computing” (Ted Neward)



Network Issues

- First law of networking:
 - Minimize round trips
- Second law of networking:
 - Avoid sending redundant data
- But what's the most important law?



Zeroth Law Of Networking

- 20 minute drive, or a trip to Pluto?
- Zeroth Law: **DON'T**
(...OK, Don't *without a reason*)
- Reasons:
 - You need to access a remote resource
 - You need capacity and/or availability (clustering)
 - Your vendor's marketing department says it's cool



EJB And The Zeroth Law

- EJB 1, everything was remote
 - (To support clustering, for sure)
- Hard knocks introduces:
 - Session Facade
 - Make coarse-grained business methods that support the use of the data, not the OO design
 - “Composite Entity”
 - Avoiding the network becomes so important we avoid using entities when using entities!



EJB And The Zeroth Law

- Then, in EJB 2, we get local interfaces
 - And quite soon most containers allow local communications between web container and ejb container
- Now in EJB 3, there's hardly even the expectation of remote access to entities
 - We'd probably serialize them and copy them in preference



RMI And The First Law

- Prior to RMI, most distributed computing systems required you to design the “structure” that would carry data over the network
 - RMI changed this, allowing almost any object to be used
- RMI and EJB users quickly invent:
 - Session facade — coarse-grained methods
 - Transfer object — get all the data in one place





RMI And The First Law

- In EJB 3, we often suggest serializing entities directly rather than using custom designed transfer objects
 - Is that always a good idea?
- How about using transient, or customized serialization?
 - OO tells us we don't need to understand the innards of an object
 - How well would invisible-inconsistent-object-semantics sit with this notion?



Web Services & SOA

- XML is greedy
 - Parsing & generation of XML is computationally very demanding
 - Adding this to network latency is bad
 - Network bandwidth overhead grows hugely too
- They told us “it's plain text which is good”
 - But now, everyone has to compress it anyway



Transactions Over Networks

- Transactions limit scalability
 - The longer the transaction duration, the worse the potential impact
- Networks are slow
- Networks are unreliable
- Consider the consequences of transactions spanning networks
 - Potentially much longer duration
 - Partial failure forces timeout (and rollback)



Memory Effects

- Reference following garbage collection (*e.g.* Java) is incompatible with large virtual memory
 - GC breaks the optimization on which virtual memory is based
 - Do not let the VM allocation exceed (90% of) the physical memory available



Memory Effects

- EJB introduces application-level virtual memory (HTTPSession and Stateful Session Beans)
- Naively implemented, these would increase GC load
 - Page out operation would waste an object, page in would have to allocate a new one
- Implementation along with pooling would seem to correct this



Early Java GC

- In early Java releases (1.0, 1.1) GC was slow and costly
 - Many people, including the EJB team, adopted pooling approach to mitigate this
 - Reuse the object, rather than collect one and allocate another
 - In 1.0 and 1.1 this was unequivocally a good thing
- Since 1.2, GC has become ever faster



Modern Java GC

- GC today is faster than you can code a simple pool
 - *i.e.* Code for get and put operations is more time consuming than typical GC
 - Still worth pooling if you have costly, but reusable object initialization
 - Still worth pooling non-memory artifacts, notably DB connections
 - Sometimes, objects should be pooled for memory use



Modern GC Weakness

- Unfortunately, the new GC algorithms are an optimization, based on an assumption
 - Short-lived objects, and long lived objects
 - But objects of medium lifespan are not collected efficiently
 - *e.g.* conversational state



Your Pool Has A Leak

- Unfortunately, most pools don't really work anyway
 - What's in an object?
 - So, what's pooled?
 - And what's not?
- How can you create an effective pool?
 - `final` variables
 - mutable component objects
 - And no luck with `String`!



The Power Of Decoupling

- Messaging provides spatial and temporal decoupling
 - Along with stateless services provides for capacity and redundancy
- XML provides syntactic decoupling
 - Can “easily” change the message format
- But XML does not provide semantic decoupling
 - So don't expect to be able to connect “just anything” just because they're “services”



Summary

- Transactions are expensive, and can place a hard-limit on scalability
 - Trying to generalize them might be unacceptably inefficient
 - Spreading them over a network is bad too
- Good DB behavior depends on understanding how the data are used
 - Which usually means that the programmer has some important work to do



Summary

- Networks are slow and unreliable
 - If your network is fast, your CPU is *way* faster
 - Three laws, starting with the zeroth
- Java's memory handling is good, but has some vicious failure modes
 - Make sure you're fixing the right problem, not the one you think might be there
 - Beware of other folk's attempts at general solutions too





The Cost Of New Technology

Simon Roberts

simon@dancingcloudservices.com

