Toward Automatic State Management for Dynamic Web Services

Jeff Chase
Department of Computer Science
Duke University
chase@cs.duke.edu

Amin Vahdat
Geoff Berry
Landon Cox
Geoff Cohen
Michael Dean

Scaling Internet Services

“The Internet is growing exponentially”
	etc., etc....

With 100M+ users out there, popularity is something for Website builders to fear.
Scalability in the Small

Internet sites can achieve scalability in the small using clustering, bigger machines, and fatter pipes.

Drawbacks of serving from a single network site:
- vulnerable to site failure
- higher latency and
- communication cost

Scalability in the Large

One way to achieve scalability in the large is to push the service out into the network, closer to the clients.

Wide-area caching and replication promise more available and responsive services.

Web cache vendors: Inktomi, Novell, CacheFlow, NetApp
Distributed caches: NLANR Cache Infrastructure (Squid)
Replicated Web hosting providers: Akamai, Sandpiper
The Trouble with Dynamic Content

Dynamic documents are produced by code (e.g., cgi) executing over service state, e.g., materialized from a database or other external repository.

Problem: Existing frameworks for Web caching and replication do not handle dynamic content.

Why Dynamic Content Is So Important

Dynamic content is a key aspect of the present and future Web.

- Web servers become “Web application servers”.
  - personalized content presentation (my.*.com)
  - Web-based mail, commerce, finance, medicine, etc.
  - interactive services for storage/retrieval/presentation
- Application Service Providers use the Web as a delivery vehicle for “futz-free” applications.
  - no installation, no upgrade, no backups, no mess, no fuss
  - easy access from diverse platforms (e.g., mobile)
  - “apps on tap”
Scaling Services with Dynamic Content

Solution: cache/replicate the service itself (code and data) instead of the documents it generates.

Issues

- mechanism for migrating code/data
- replica placement
- request routing
  * state management/consistency
  - security
  - resource management

Web Application Proxies extend static Web proxies to cache and execute service code and data (service caching).

Toward Automatic State Management

Our objective is to facilitate caching and replication of dynamic content.

1. Consider a growing class of services built using server-side Java technology.
   
   Leverage Java’s transportable code and data.
   
   JavaServer Pages (JSPs) add useful constraint to Java’s servlets.

2. Focus on the subproblem of state management.
   
   Internal service state must be consistent and current.

   
   Can we transparently convert unscalable service implementations into scalable ones?
Why “Toward”? Are We There Yet?

We simplified the problem to make it tractable:

- **Heterogeneity of state**: we handle Java objects only.
  Materialize data from external sources as Java data structures.

- **Concurrency control**: a hard problem, so we use brute force.
  Prototype uses “single shot” reader/writer locking on object groups.
  Updates originating at different replicas must be nonconflicting.
  Service programmer must help identify points when state is internally consistent (commit points).

“Consistent” class interfaces

Our current solution offers semi-automatic state management for “well-behaved” Java services.

---

**Portrait of a JSP Web Application**

For our purposes, a “Web application” is a cloud of JSP servlets and objects.
The Project in a Nutshell

1. Use bytecode transformation to rewrite the service code.
   JOIE (J* Object Instrumentation Environment) is a toolkit for building bytecode rewriters: it’s “ATOM for Java”.

2. Inject calls to a simple caching/replication package (Ivory).
   “Shasta or Midway for the Web”

3. Use an incremental variant of Java’s Object Serialization framework to propagate state among replicas.
   “Rsync for JavaServers”

4. Illustrate with a minimal caching/replication framework.
   Extend conventional Web proxy caching with service caching in Web application proxies.

The Role of Bytecode Transformation

- **Servlet**
- **AutoWriter**
- **CaptureWrites**
- **SpliceCommit**

- Consistent action entry points implement **Consistent** interface.
- Class-specific object serialization methods
- Write barriers
- Prologue and epilogue for **Consistent** methods
The Transformed Service

Transform all servlet classes and associated object classes in the “cloud”.

Servicing a Replica (Pull)

On a name lookup miss, or when proxy expiration time expires, pull missing objects and updates from primary server.
Meeting the Goal of Simplicity

A key objective is to avoid (re)implementing a “full-blown” distributed object system.

- *There is currently no reference faulting mechanism.*
  - Leverage JSP symbolic naming scheme.
  - The granularity of fetch is the closure of a named object.
- *There is no synchronous update/invalidation mechanism.*
  - Each replica sees a self-consistent state...but it may be stale.
  - Updates propagate all modified objects in the shared view.
- *Represent references as OIDs, but only on the wire.*

Managing Multiple Replicas

The state manager tracks object membership in each replica’s view.

A serializer propagates objects (and their closures) to views incrementally.

The serializer uses (and updates) OID mappings in the view tables.
What Does It Cost?

proxies and servers
Sun Ultra 140
167 MHz UltraSPARC
128 MB RAM

workload
toy portal emulation
small data (~300 KB)
2500 objects
(stocks, news, etc.)
3KB response
demand*5 users
random profiles

15% update per second
2-second proxy refresh times

Scalability Benefits: A Small Experiment

median response
times below
saturation:
~205 ms

15% update every 5 seconds
2-second proxy refresh times
Conclusion

1. Replication of dynamic Web services is a worthy challenge.
2. In the Java environment, bytecode transformation is a powerful tool for automating replica state management.
3. The prototype enables service caching for a class of Java-based dynamic services, improving scalability.
   - modest state demands with minimal write sharing
   - leverage JSP-style symbolic naming for partial replication (caching)
   - Web application proxies extend the benefits of static Web caching
4. Web applications must be “well-behaved” to benefit.
   - Key research questions concern the contract between the Web application and the replication service — and how to enforce it.

Performance Cost of Write Barriers

![Performance Cost of Write Barriers Graph](image-url)