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

Toward Scalable Wide-Area Services

1. The Internet is growing exponentially, etc., etc.
   - With 100M+ users out there, popularity is something for Website builders to fear.
2. Internet sites can achieve scalability *in the small* using clustering, bigger machines, and fatter pipes.
3. However, centralizing a service at a single network site leads to:
   - single point of failure
   - higher overall communication cost
   - higher network latency

Web Caching and Replication

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

- Web caching is now a thriving market.
  - Inktomi, Novell, CacheFlow, Network Appliance
  - NLANR National Caching Infrastructure
- Emerging Web hosting providers offer replication services.
  - Akamai, Sandpiper, etc.

Wide-area caching and replication promise better availability and better response times than single-site solutions.

The Trouble with Dynamic Content

Dynamically generated content presents key challenges for caching and replication frameworks.

- Dynamic documents are not cacheable in current Web proxy caches: they may change at any time.

We must replicate the service itself (its *code* and *data*) instead of just the documents it generates. This creates challenges:

- Internal state may have strong consistency constraints.
- Raises security and resource management issues.
- What to replicate and where? How to route requests among replicas?

Why Dynamic Content Is So Important

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

- Web servers become “Web application servers”.

  portals and personalized content (*my.*.com*)

  online commerce, auctions, Web-based mail, etc.

- Dynamic content allows use of the Web as a delivery vehicle for “futz-free” applications.

  no installation, no upgrade, no backups, no mess, no fuss

- In the future: Web-based applications (and storage) offered as a value-added service by broadband ISPs.

  benefits include easy mobile access

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.

3. Goal: make state management *automatic*.

    Can we transparently convert unscalable service implementations into scalable ones?

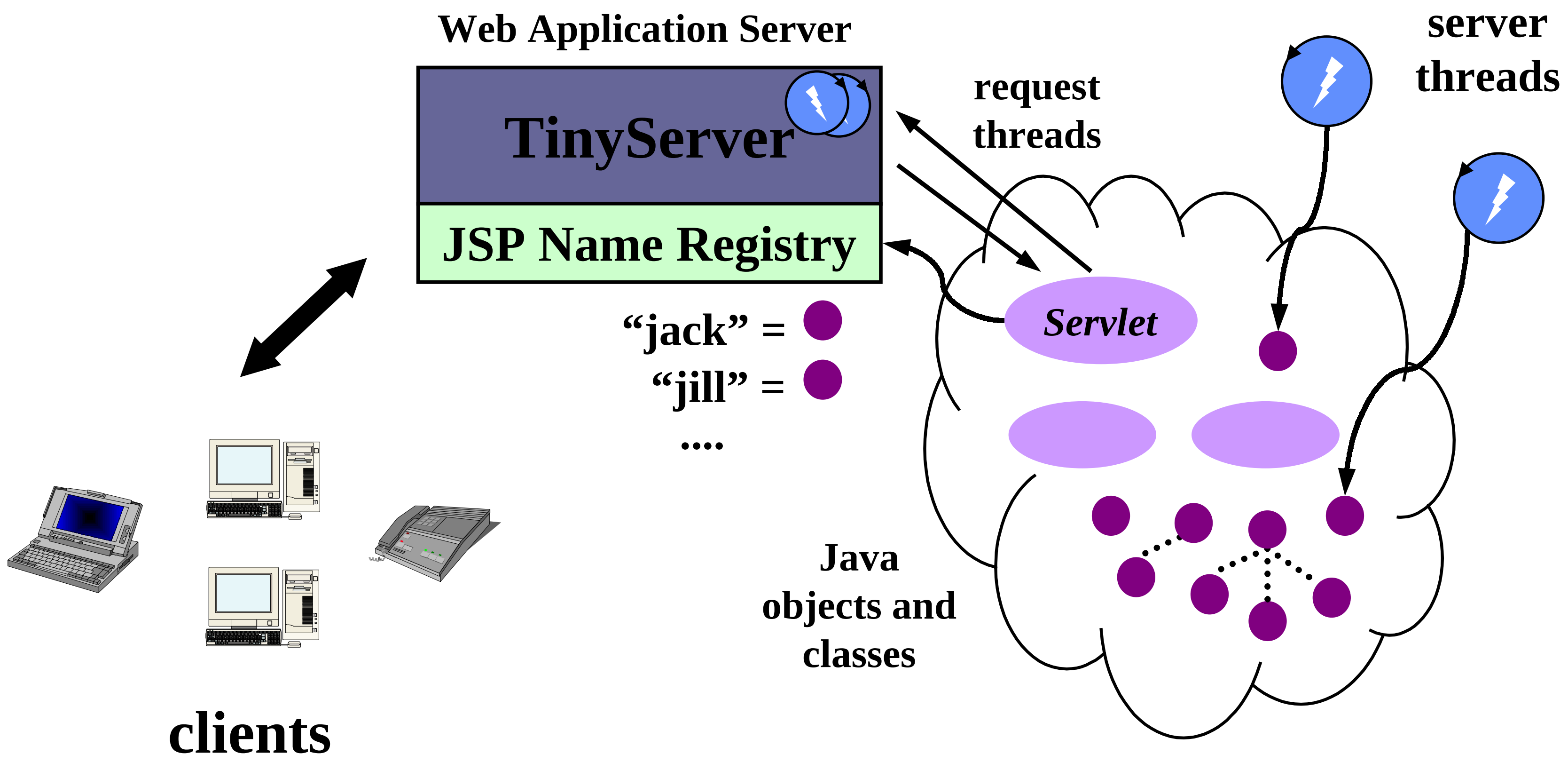
Why “Toward”? Are We There Yet?

We simplified the problem to make it tractable:

- Heterogenity 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 coarse-grained “single shot” reader/writer locking.
  - Updates originating at different replicas must be nonconflicting.
  - Service programmer must help identify consistent commit points.
    - “***Consistent***” class interfaces

Our current solution offers semi-automatic *dissemination* of dynamic content 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*** **O**bject **I**nstrumentation **E**nvironment) is a toolkit for building bytecode rewriters: it’s “ATOM for Java”.

2. Inject calls to a simple caching/replication package (*Ivory*).

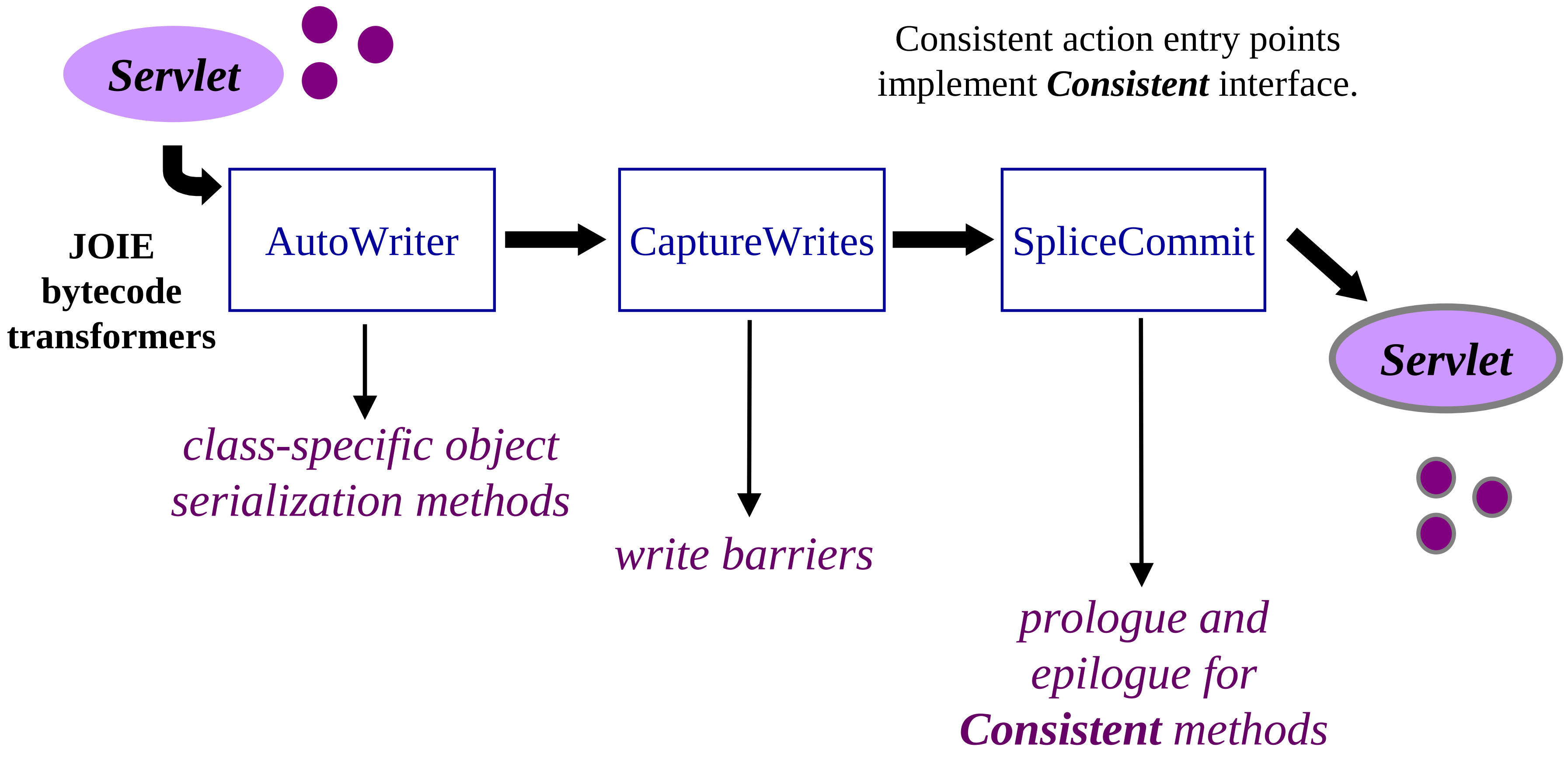
   “Shasta for the Web”

3. Use an *incremental* variant of Java’s Object Serialization framework to propagate state among replicas (*pickles*).

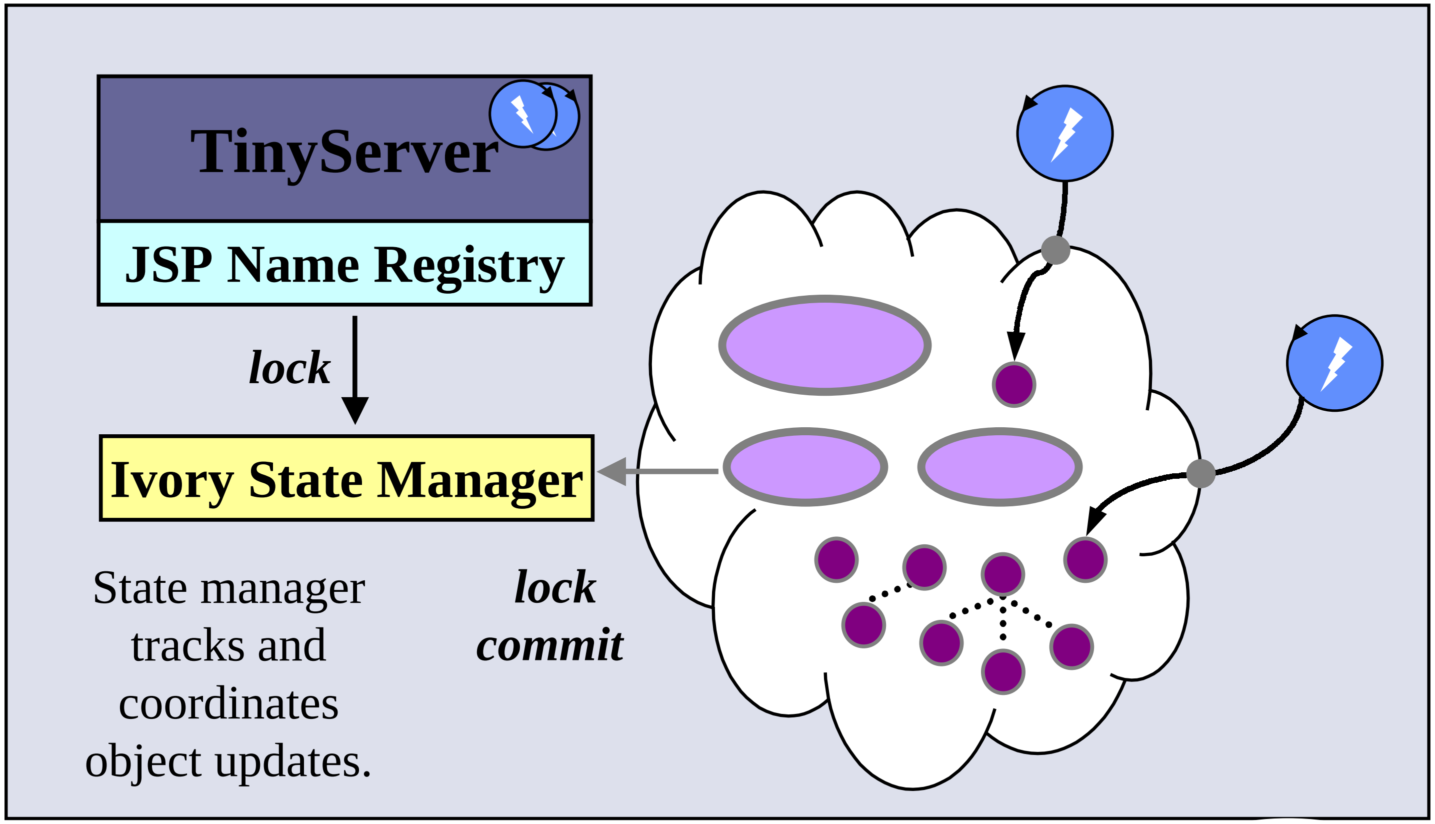
4. Illustrate with a minimal caching/replication framework.

   Extend conventional Web proxy caching with *service caching* in *Web application proxies.*

The Role of Bytecode Transformation


Servlet
Consistent action entry points implement ***Consistent*** interface.
JOIE bytecode transformers
AutoWriter
CaptureWrites
SpliceCommit
Servlet
class-specific object serialization methods
write barriers
*prologue and epilogue for **Consistent** methods*

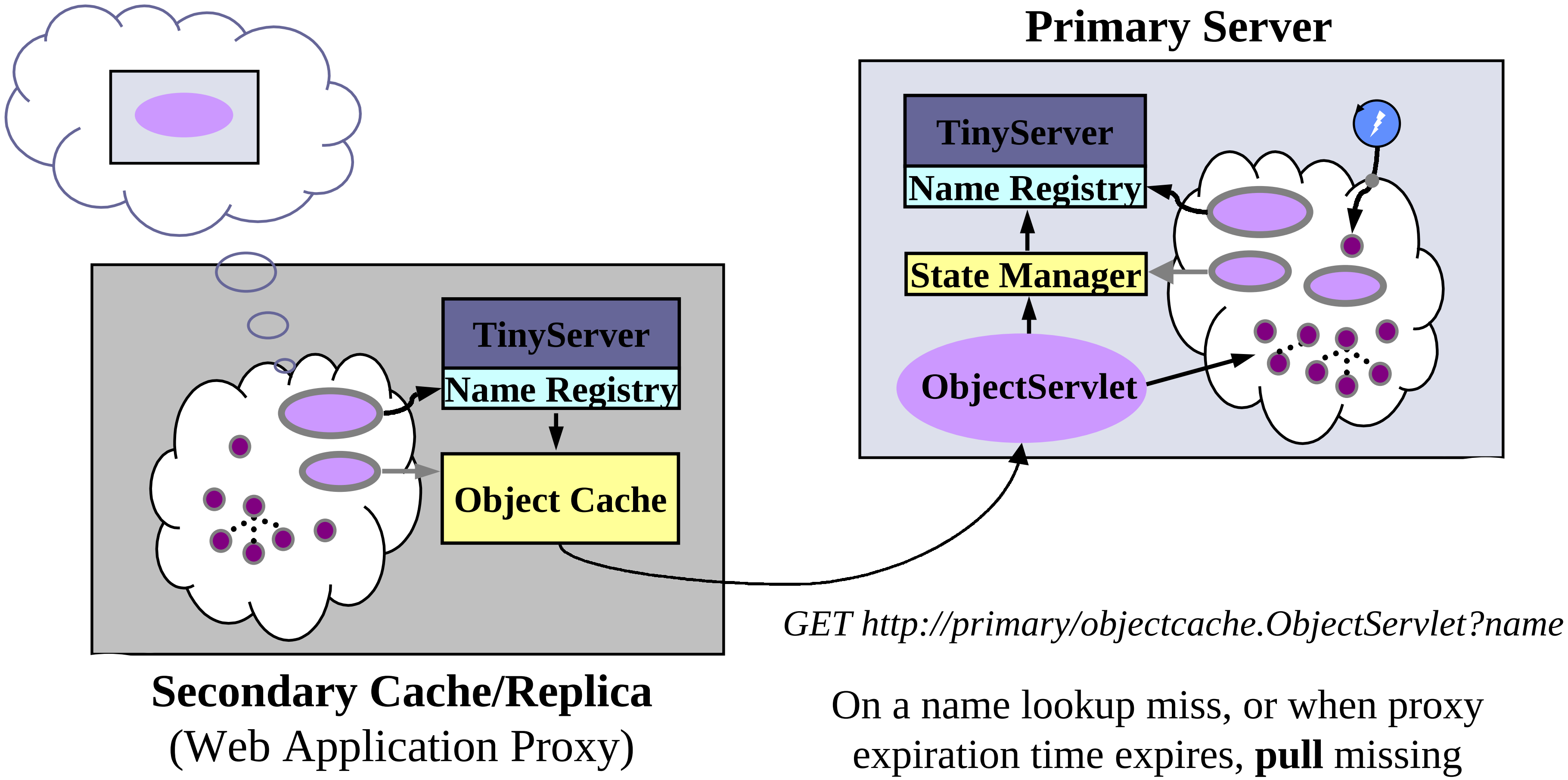
The Transformed Service



State manager tracks and coordinates object updates.

Transform all servlet classes and associated object classes in the "cloud".

Servicing a Replica (Pull)



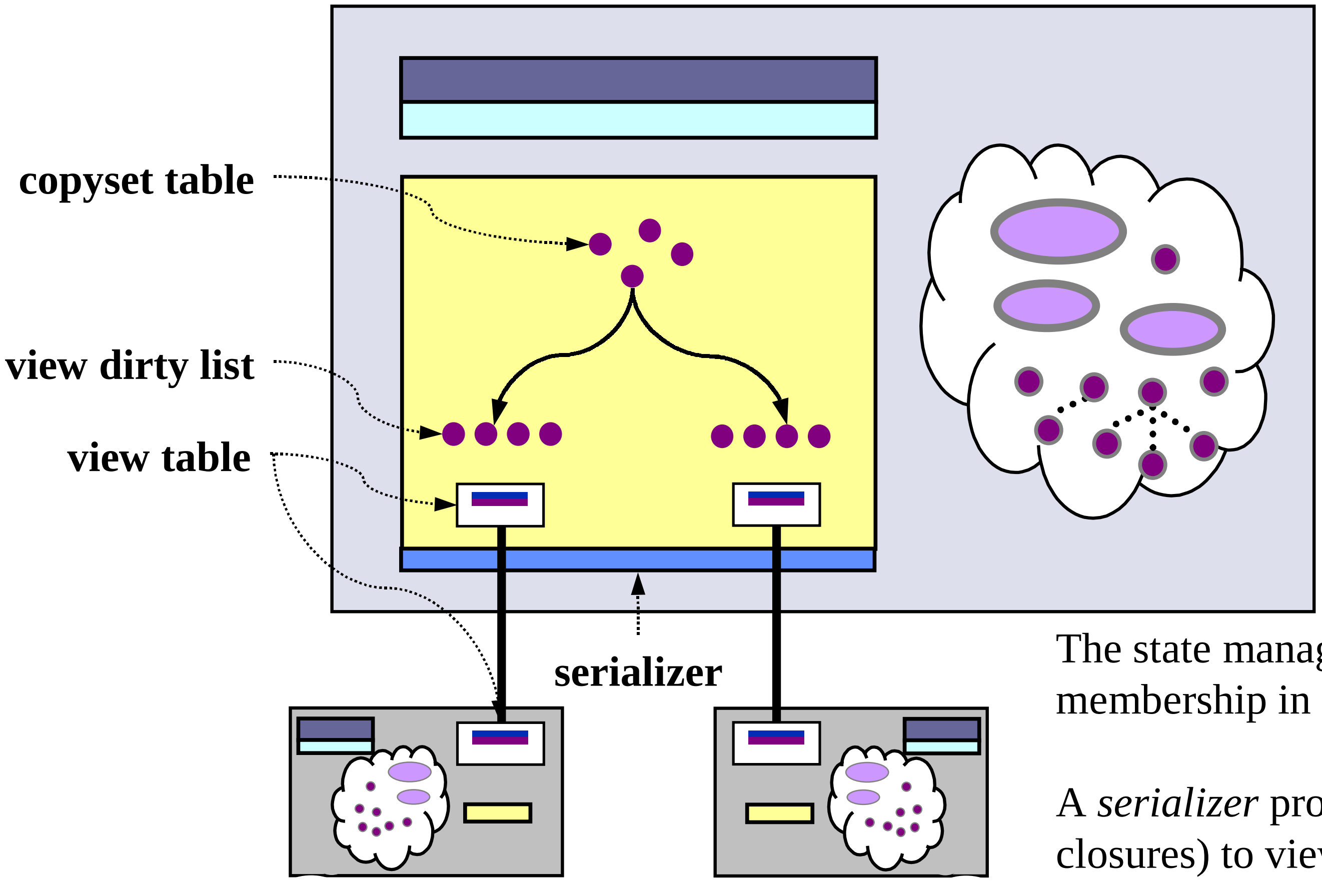
On a name lookup miss, or when proxy expiration time expires, **pull** missing objects and updates from primary server.

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 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



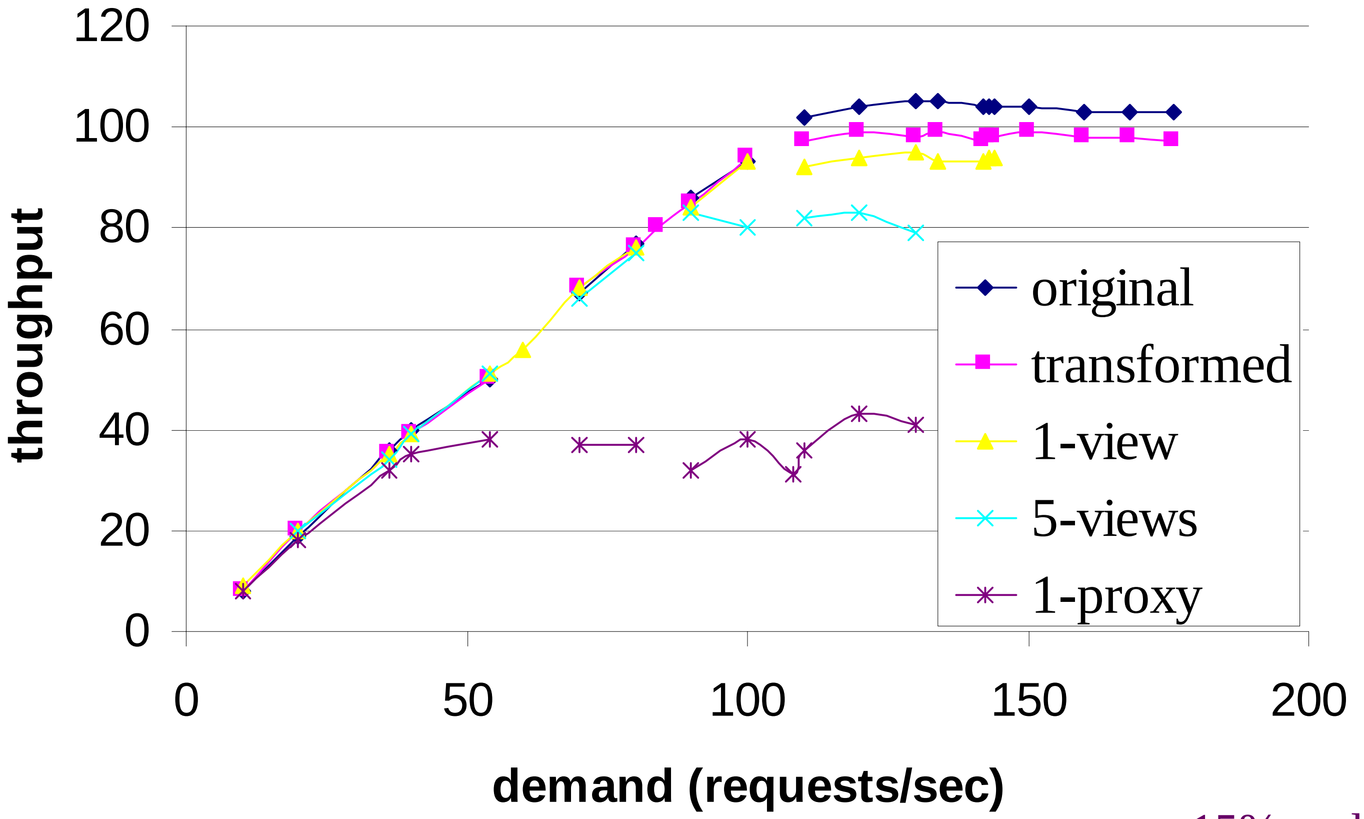
requires
O(*mn*) space

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 All This Cost?



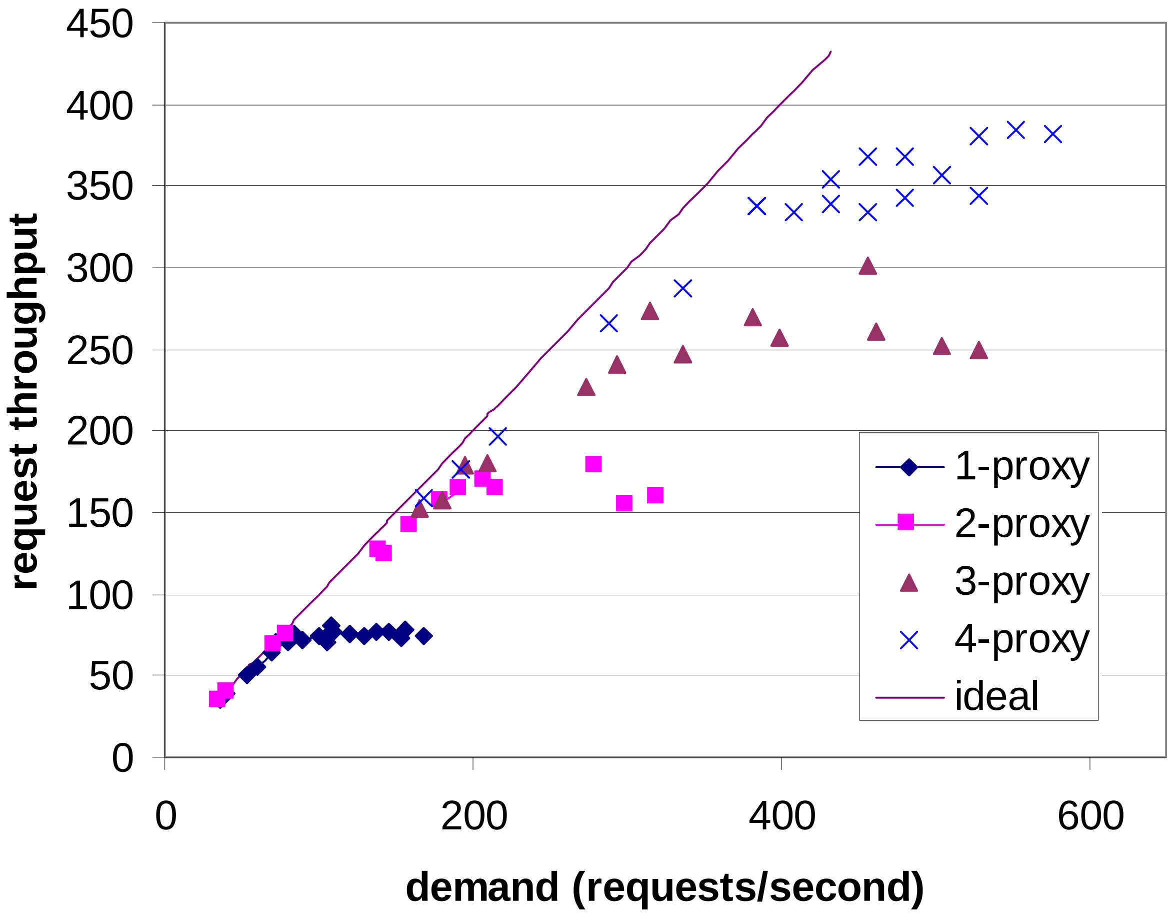
proxies and servers
Sun Ultra 140
167 MHz UltraSPARC
128 MB RAM

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

DUKE Computer Science

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. Concurrency control is the key difficulty.

Some Limitations

1. Coarse concurrency control can create locking bottlenecks.

   services with interdependent requests or I/O in writer threads

2. Bug: object tables inhibit collection of replicated objects.

   may require enhancements to Java environment

3. Replicas are vulnerable to stalled or broken TCP connections with peers during update propagation.

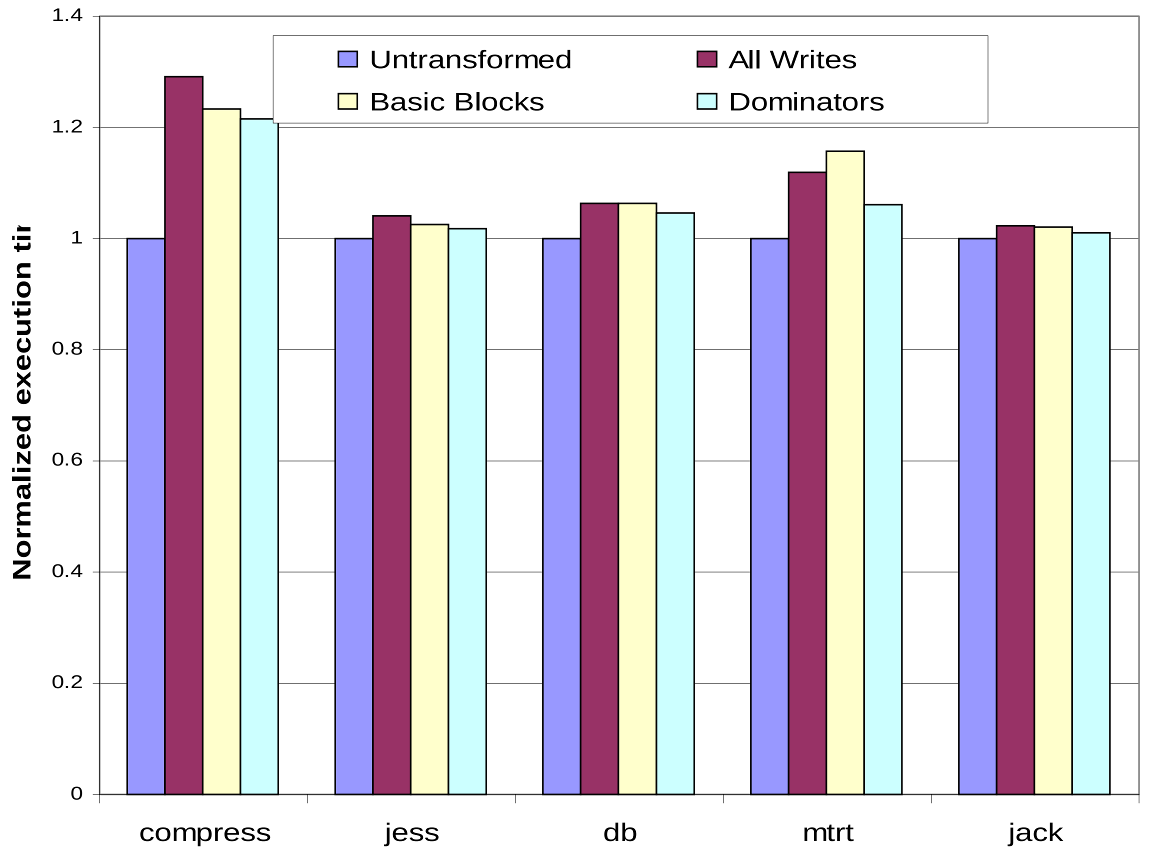
4. This work does not address security and resource management issues.

5. We use simplistic policies for replica creation.

   name-based caching in the service caching framework

DUKE Computer Science

Performance Cost of Write Barriers


Untransformed
All Writes
Basic Blocks
Dominators
Normalized execution ti
compress
jess
db
mtrt
jack