A Case for Fluid Replication

Brian Noble, Ben Fleis, Minkyong Kim
University of Michigan
The Problem: Variable Performance

As systems scale, performance becomes unpredictable. Unfortunately, people like predictability.

Three reasons for variance:

- Bursty demand for end-services
  - Someone posts a pointer to you on slashdot

- Bursty demand for network resources
  - Congestion between you and your service

- Mobile workforce means rapid topology changes
  - Suddenly far from your service in network terms

Rates of each of these increasing: can’t manage by hand
Current Approaches Fall Short

Cluster-based replication
- host services on cluster
- recruit cluster resources when load increases
- copes with variation in end-service demand
- doesn’t address network/mobility

Peer-to-peer replication
- cache data, operate on it locally
- exchange updates with peers
- addresses variable performance

Peer replication introduces other problems
- clients are resource-limited, unsafe compared to servers
- cannot bound convergence of updates
Fluid Replication: Best of Both Worlds

Retain notion of service replicas
  safety, consistency, ease of administration

Allow those replicas to be created anywhere
dynamically instantiated by clients
responding to changing demands on resources
stretch the bonds of cluster-based replicas

Key abstraction: the WayStation
managed by local administration domain
provides services to local, visiting users
forms loose confederation of cooperative nodes
address mobility or demand-induced network costs
Do These Networking Costs Matter?

Compare simple compilation over NFS
client connected to server via router

Router runs trace modulation
vary latency and bandwidth

Changes have significant impact
  latency increase from negligible to 20ms: 5x worse
  bandwidth decrease from 10Mb to 100Kb: 1.5x worse
  degrade both latency and bandwidth: 5.5x worse

Admittedly a pessimistic example
  NFS uses many short, synchronous messages
Technical Challenges

Measure and react to networking costs especially difficult over wide-area

Finding a WayStation to use must be close to (mobile) client

Managing consistency between replicas
   WayStation is close to client, far from service
   key to providing bounded convergence

Moving from replica to replica
   clients have strong expectations of consistency

Doing all of this safely and securely
Generating Connectivity Estimates

Monitor activity between client and services
  passive observation, avoid additional congestion
  measure request/response timestamps, sizes
  isolate network, report service time in response

These give spot observations of latency, bandwidth
  adjacent request/response pairs of different sizes
  these vary wildly in mobile, wide-area networks

Apply filtering to generate estimates, error bounds
  filter must detect changes quickly (agility)
  filter must smooth unimportant changes (stability)
Filtering for Agility, Stability

Borrow techniques from controls, signals

Start with simple low-pass filter, similar to TCP round-trip time
new estimate = G(this observation) + (1-G)(old estimate)
constant, high gain gives agility without stability
constant, low gain gives stability without agility

Intuition: adjust gain to select for one or the other
increase gain (agile) when observations are stable
report both estimate and confidence in it (stable)

Early experience suggests this approach will work
can lose stability, but is reflected in confidence
Finding WayStations

When a client discovers performance is poor/turbulent
must find a WayStation to hold replica
must be close enough to be useful
particularly hard for mobile clients

Client discovers nearby WayStation through distance routing
routers estimate performance to neighbors

Distance-based discovery uses this information
broadcast with a cost limit
prune at routers if exceeds cost
WayStations return network costs, load information

Lazily populate replica on chosen WayStation
Consistency Maintenance

Service coordinates between itself and each WayStation replica peer-to-peer systems calls this the “star topology”

Managing consistency of each WayStation critical replicated when client far from service close to WayStation WayStation probably far from service

WayStation+clients: island of good performance careless management eliminates those gains

Two dimensions along which consistency schemes described strength of guarantee: what clients can assume frequency of maintenance: how often guarantees enforced
**Strengths of Guarantee**

**Last-writer: no guarantees**
- each replica can update independently
- updates logged, periodically exchanged
- if updated in two places, keep only one

**Optimism: guaranteed detection of conflicts**
- update independently, log and exchange
- service checks for serializable operations
- safe operations applied, unsafe flagged as conflicts

**Pessimism: guaranteed prevention of conflicts**
- require replicas to obtain exclusive access before each write
- can perform adequately if high write-locality
Frequency of Guarantee

Each WayStation is a replica of some service
  pessimistic: interacts with service each write
  optimistic, last-writer: periodic interactions

Service manages all WayStation replicas
  updates converge in 2x longest period

How to set this interval properly?
  poor WayStation/service connectivity: longer
  higher update rates, tighter convergence: shorter

We are only beginning to grapple with this question

Service can become bottleneck: need cluster-based replicas
Selection of Consistency Scheme

Service provides default scheme for most clients
- publish-subscribe, mirror: last-writer is fine
- workloads with very high write locality: optimism
- workloads with fine-grained write sharing: pessimism

Service and WayStation monitoring informs frequency
- may require an upper bound for some applications

Clients may choose to upgrade/downgrade scheme
- application enters a region of fine-grained interaction
- client unwilling to pay performance penalty

We must arbitrate between conflicting schemes
- strongest guarantee wins, place burden where acceptable
Migrating Clients Expect Strong Guarantees

A client expects its writes to be persistent session guarantee: “read-your-writes” even when migrating between replicas not provided by last-writer, optimism

Worst case: synchronous flush client declares intent to migrate WayStation flushes all updates to service client then free to move expensive, since WayStation and service are far apart

Three optimizations are possible
Migration Optimizations

Client has some updates in its cache, forms a postfix of the update log, can replay those to new WayStation.

New WayStation may be closer than service, can apply path compression, forward updates directly.

Use consistency promotion to defer operations, client requests promotion to strict, invalidate updated objects at service, propagate asynchronously, preserves order at expense of eventual transfer.
Doing This Securely

WayStations are administered by local domain
provide services to foreign users
each party suspicious of the other

Establishing trust in advance not practical
exposes seams in the WayStation infrastructure
won’t scale to large deployments

Apply paths/hierarchies of trust
  can we deal with the dilution problem?

Can we defer judgements of trust?
  what can be deferred, how can it be done efficiently?
Related Work

Fluid replication borrows ideas from many places

- Grapevine: first use of replication with weak consistency
- Cluster-based replication: Challenger, Fox, Pai
- Peer-to-peer systems: Ficus, Bayou
- Extensible DSM: Munin, Khazana
- Optimism (for mobility): Coda, Ficus, Bayou
- Network Prediction: NWS, Lai, SPAND

In addition, other systems can provide components

- WebOS: mechanisms to build systems like FR
- CRISIS, PKI: cross-domain authentication
Conclusion

Variation in performance of distributed systems getting worse with scale, mobility

Fluid replication: cope with this variation safety, bounded convergence of server-based approaches performance, efficacy of peer-to-peer systems

This work is just beginning very interested in feedback