When Local Becomes Global

An application study of consistency in a networked world

Erik Riedel, Susan Spence, Alistair Veitch

Hewlett-Packard Labs

Hewlett-Packard Laboratories

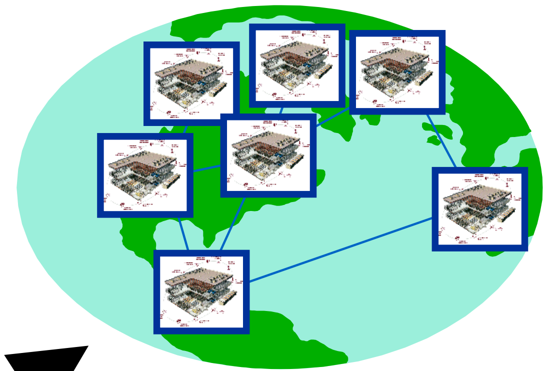
Outline

- Global Data Placement
- Existing research landscape
- Data consistency
- Trace analysis
- Future directions

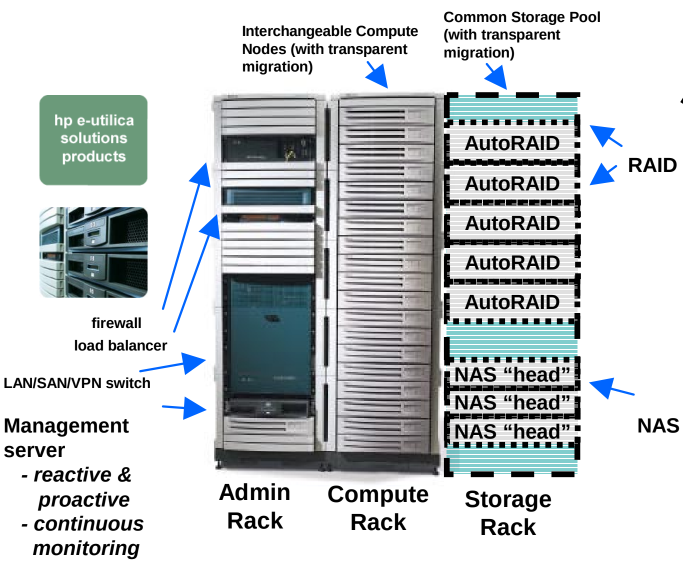
Evolution of data centers

Global data centers

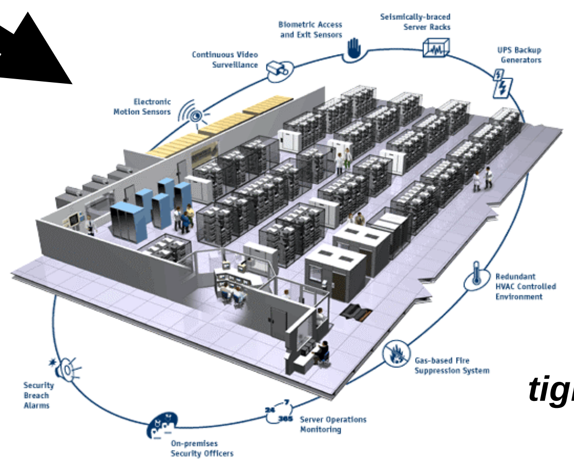
- Today - racks of compute and storage nodes, automatically managed
- Tomorrow - data distributed around the world, automatically managed



100s of data centers globally distributed



optimized racks of compute and storage

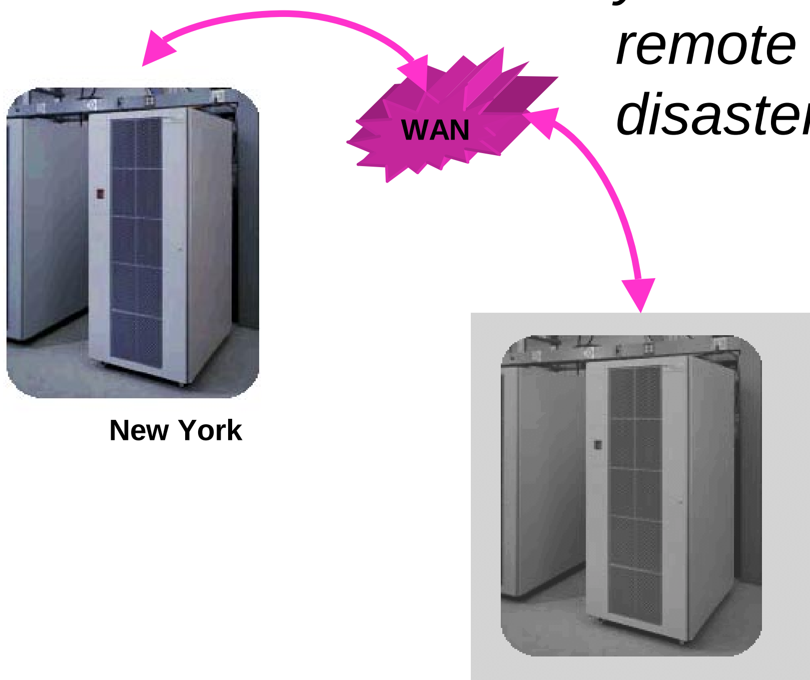


1000s of tightly packed racks

What ties them all together? The (Shared) Data

Data consistency - disaster recovery

yesterday's solution-
remote mirroring for
disaster recovery

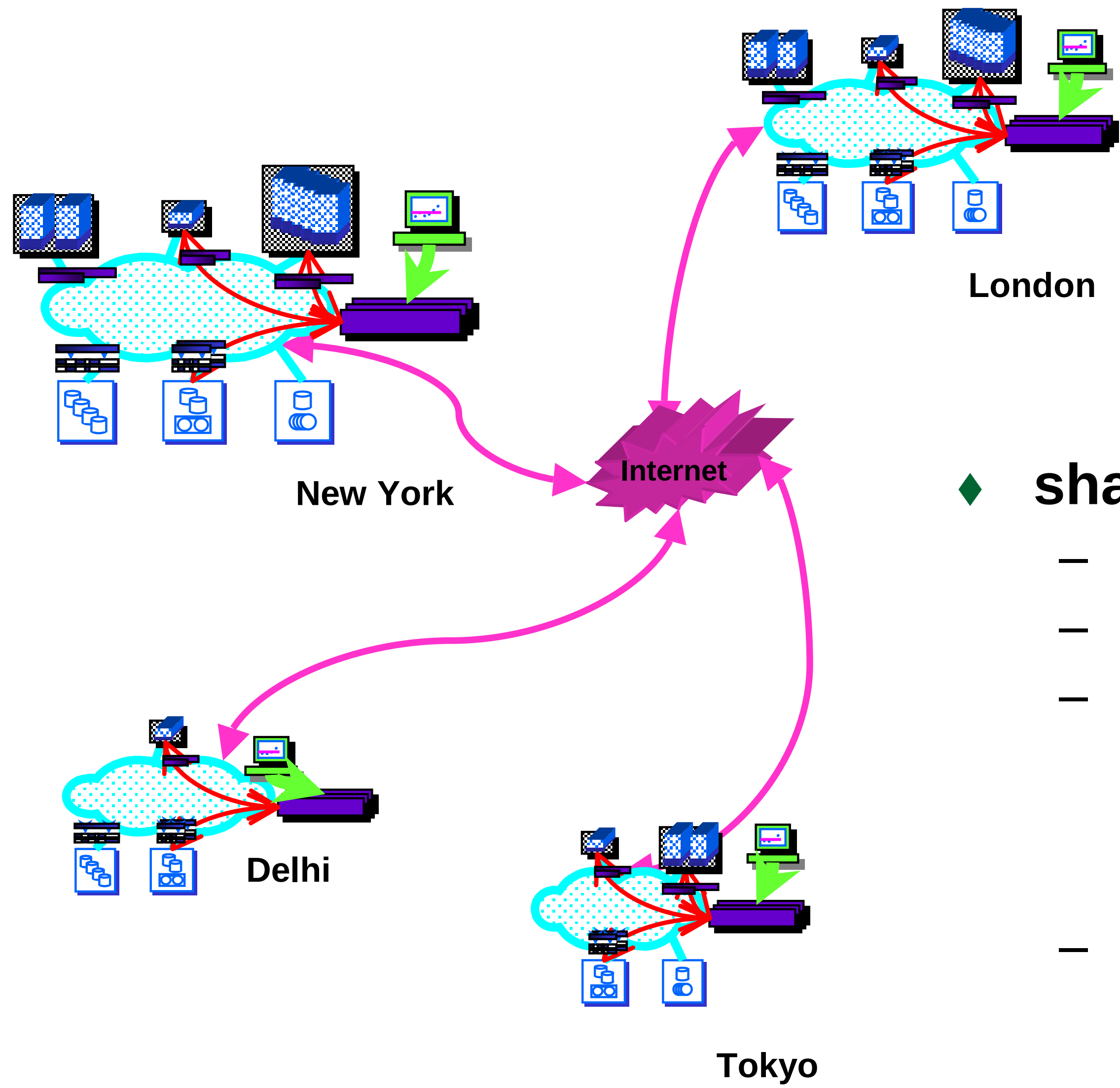


New York

New Jersey

- **remote mirroring**
  - **primary and inactive backup**
  - **limited set of semantics**
    - **synchronous,**
    - **asynchronous,**
    - **batch**
  - **primarily for disaster recovery**

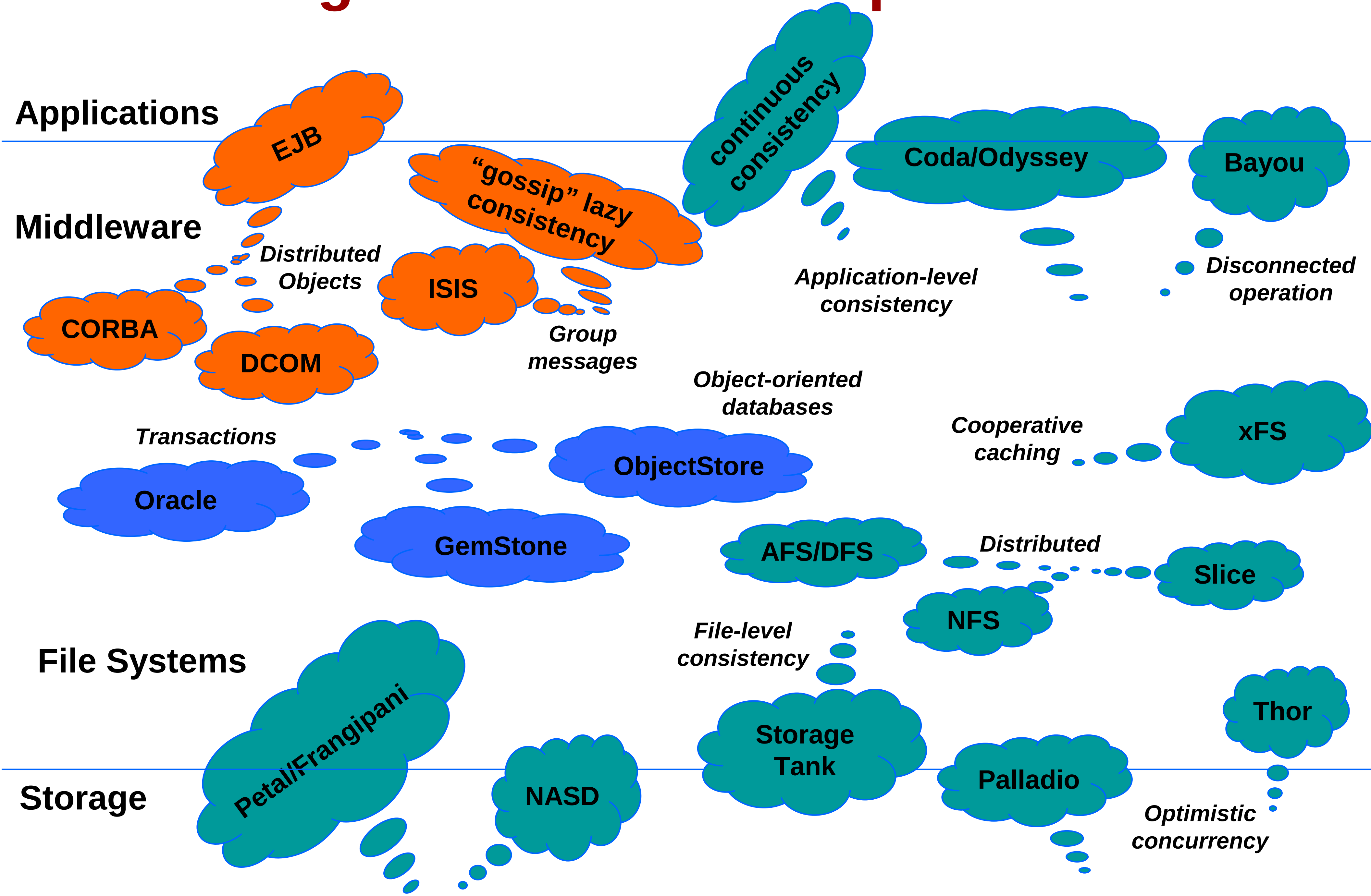
Data consistency - keeping data up-to-date



today's requirements - multiple storage islands for data sharing

- **shared storage islands**
  - ***shared* data**
  - **multiple, *active* sites**
  - **range of data semantics**
    - **web sites, email, inventory, videos, bank accounts**
  - ***adaptive* replication**

Existing research landscape



Data consistency - philosophy

Provide a general service, but take advantage of application-specific and user-specific knowledge whenever possible

- **block-level or object-level service**
  - **with application-level knowledge**
- **why not a new file system**
  - **deploying a new file system is difficult - e.g. AFS**
- **why not a new volume manager**
  - **aggregating data at volume level hides too much**

Key research challenges for GDP

- Data placement
  - what to put where
- **Data consistency**
  - **keeping the data up-to-date**
- System management and control
  - optimizing resources
- Quality of service
  - ensuring “good” service
- Security
  - protecting the data
- Global namespace
  - naming & locating the data

Potential usage scenarios

- Akamai - web content
  - static and streaming content for 3,000 web sites
  - 150,000 requests/second
  - 6,000 servers at 400 locations in 54 countries
- Wireless cell sites - content caching/prefetching
  - 100,000 sites across the U.S.
  - 100 million subscribers (~1,000 per site, ~10 active)
  - storage on a per-site basis to “follow” users
- Call centers - large U.S. consumer company
  - 17 call center sites,
  - 3,500 agents, available 24/7
  - 100 servers, shared storage
- Others
  - Cable head-ends, corporate campuses

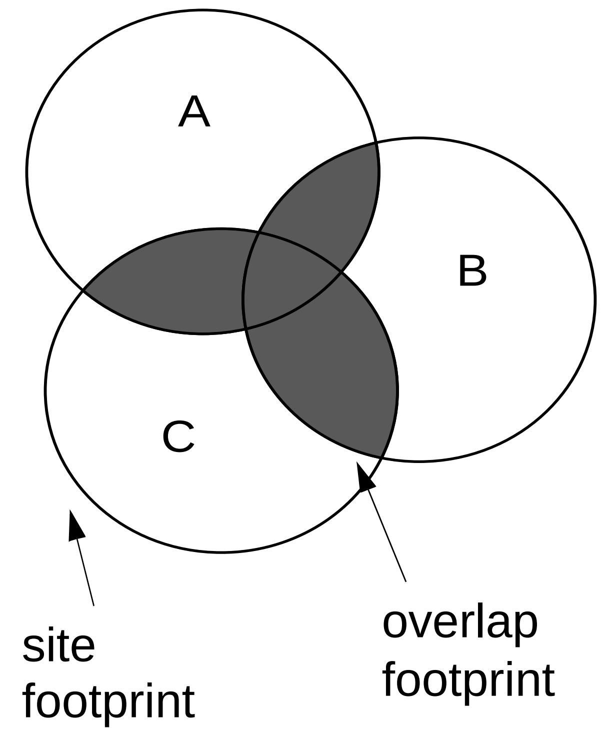
Three commercial application traces

- *cello/users* - file system workload
  - multi-user UNIX server with ~20 users, 500 GB of storage
  - divided into three sites by user - 60% / 30% / 10%
- *openmail/server* - email server
  - centralized openmail server with 3,000 users
  - divided into three sites by user - 60% / 30% / 10%
- *tpc-c/oltp* - transaction processing
  - ~120 warehouse benchmark with 50 disks
  - three identical sites operating on shared store

- back-of-the-envelope calculations for consistency
  - block-level traces, post-cache
  - footprints, inter-request data hazards - worst case

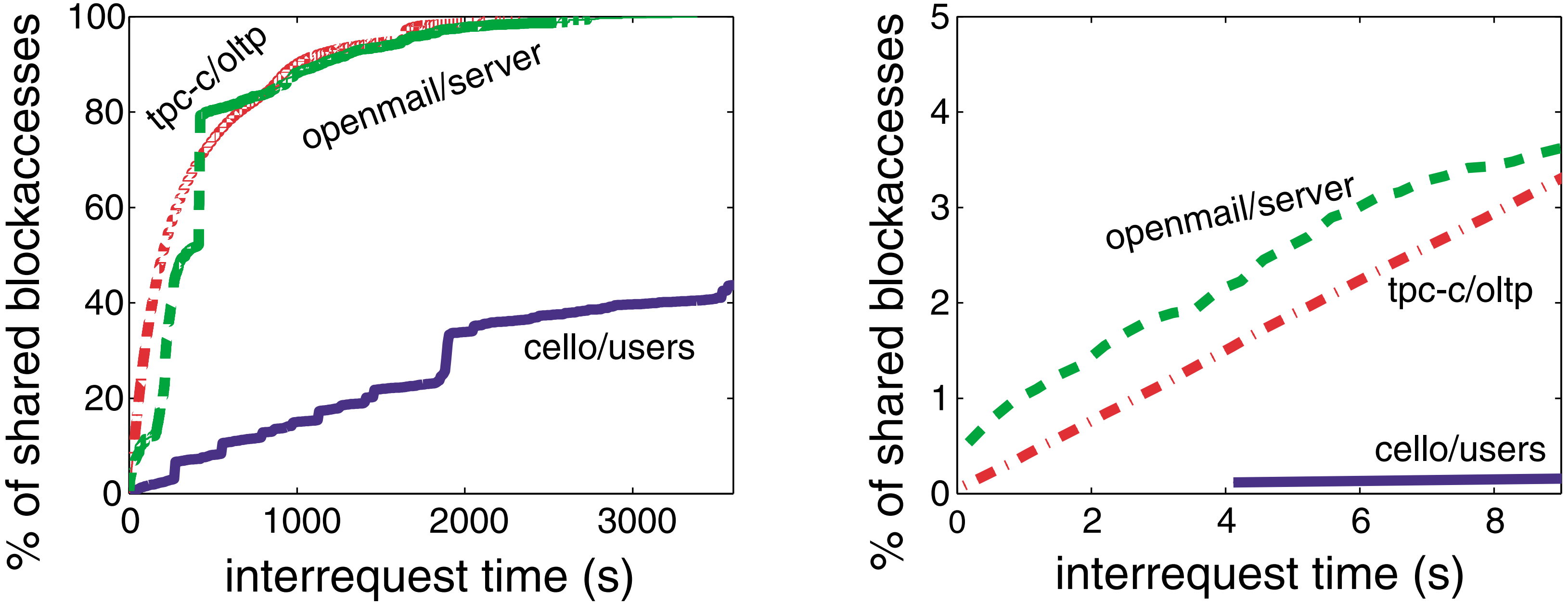
application	length	total requests (000s)	request rate (req/s)
cello/users	24 hr	1,370	380
openmail/server	1 hr	61	17
tpc-c/oltp	$^{2}/_{3}$ hr	4,220	1,620

Table 1. Amount of data moved.

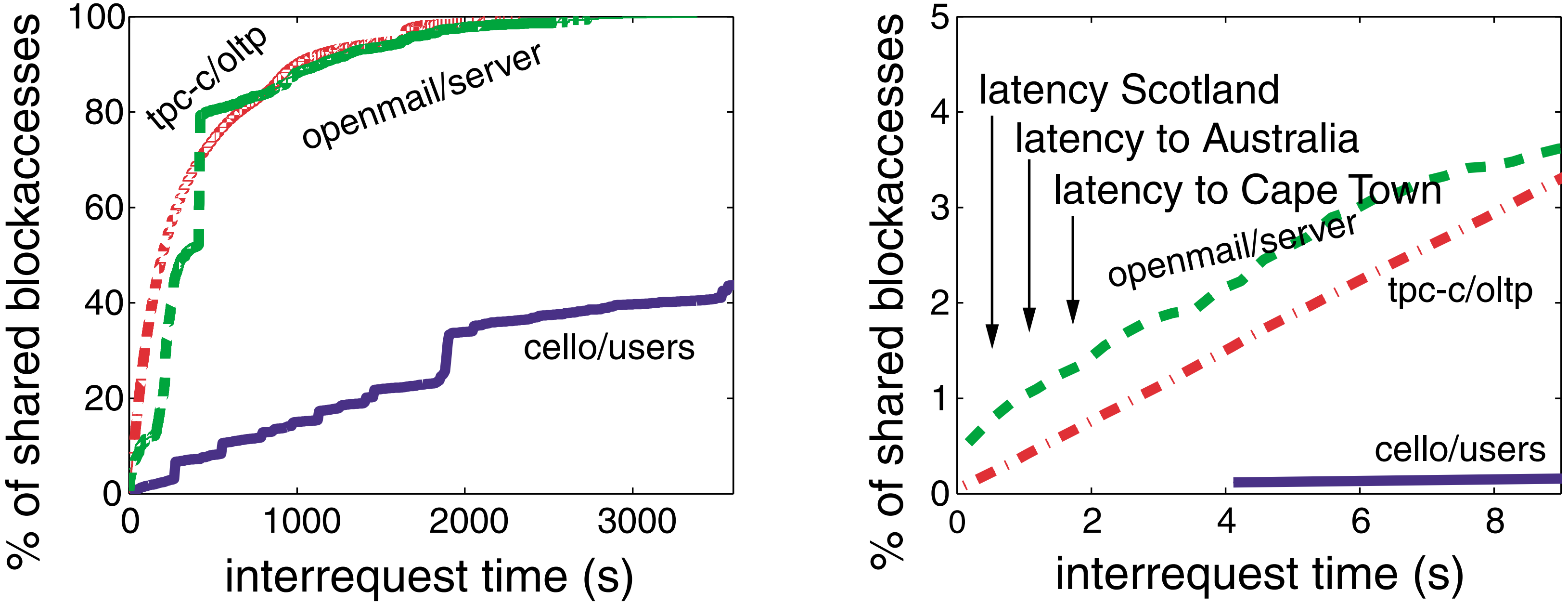


app	data read/written (MB)			site footprint (MB)				overlap footprint (MB)			
	site A	site B	site C	all	A	B	C	AB	AC	BC	ABC
cello/users	8,400	775	614	3,160	2,800	328	178	134	48	25	23
openmail/srv	211	124	23	117	75	54	13	17	1.3	1.2	1.1
tpc-c/oltp	3,410	3,340	3,100	1,700	852	950	824	555	616	533	344

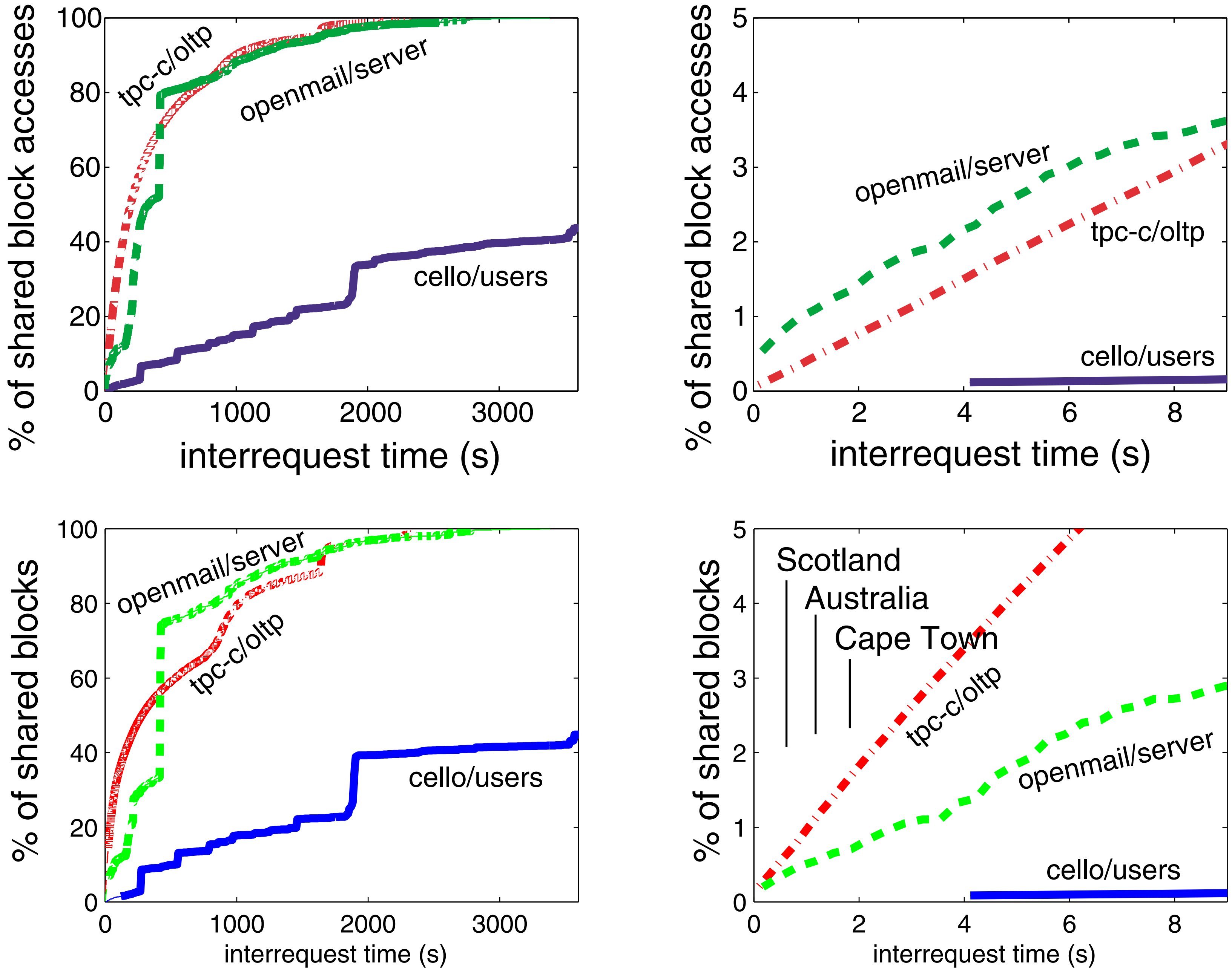
Table 1. Site footprints and overlap.



% of shared blockaccesses
interrequest time (s)
tpc-c/oltp
openmail/server
cello/users
100
80
60
40
20
0
1000
2000
3000
% of shared blockaccesses
interrequest time (s)
openmail/server
tpc-c/oltp
cello/users
5
4
3
2
1
0
2
4
6
8



% of shared blockaccesses
interrequest time (s)
tpc-c/oltp
openmail/server
cello/users
latency Scotland
latency to Australia
latency to Cape Town



% of shared block accesses
interrequest time (s)
tpc-c/oltp
openmail/server
cello/users
% of shared blocks
Scotland
Australia
Cape Town

Initial results - summary

- The good news - conventional wisdom holds
  - many workloads have little write sharing
    - except databases
  - per-site footprints are large, overlap is often low
    - so individual site replicas make sense (lots of local traffic)
    - and overlap regions can be handled specially
  - inter-request dependencies look manageable
    - “hard” consistency is required infrequently
    - points to optimistic methods, allowing occasional “mistakes”

- The bad news - challenges
  - how do you handle the “mistakes”, even occasionally
  - how to predict overlap regions in a stable way

Future directions

- Challenges
  - identify range of application requirements in more detail
    - identify a small number of archetypes
  - focus on policies informed by application behavior
    - optimistic policies when they are appropriate
      - with possible reconciliation
    - pessimistic policies when they are necessary
      - with performance penalties
  - stability of predictions
  - what to do if guess is wrong
    - reconciliation "failures" in optimistic approaches
    - must we introduce new error semantics for users?
      - where failures in distributed case are not "expected"
      - more 404 errors and Refresh buttons

Extras

Existing work

- Provide local file system semantics (more or less)
  - NFS, AFS/DFS, Petal/Frangipani, xFS, Slice
- Disconnected operation (Coda, Bayou)
  - allow updates when disconnected
  - apply application-specific merge functions to reconcile
  - may defer to the user
- Optimistic consistency (Palladio, Thor)
  - requires rollback semantics
- Multiple consistency levels (Storage Tank)
  - strong consistency w/ and w/o caching
  - publish consistency
- Continuous consistency (TACT)
  - using application-specific middleware platform

Existing work (2)

- Atomic Transactions (CORBA, DCOM, EJB)
  - Client can update data across databases in one transaction
  - Two-phase commit protocol ensures atomicity
- Synchronized code (Java)
  - Used to serialize updates by multiple threads
- Read-only/Updateable Snapshots (Distr. Oracle)
  - Snapshot synchronised with master at specified intervals
- Synch/Asynchronous Replication (Distr. Oracle)
  - Updates propagated immediately/at user-specifed intervals
  - Conflict resolution uses timestamps and site priorities
- Lazy consistency (Gossip)
- Causal message ordering (ISIS)

Consistency - web vs. file systems

- World wide web - relatively loose consistency
  - users accept broken links
  - hit “reload” as a standard response
  - endure “Web site found. Waiting for reply…”
  - willing to search and accept approximate results
  - many documents no longer exist
- Contrast to file systems - strong consistency
  - broken link considered a serious failure
  - open succeeds the first time
  - very few people search in `lost+found`
  - expect documents to be there years later