

Persistent Store Interface: A foundation for efficient, scalable persistent system design

Stephen M. Blackburn and Robin B. Stanton

Department of Computer Science

Australian National University

Canberra, Australia

<http://cs.anu.edu.au/~Steve.Blackburn>

August 1997

ACSys

**Cooperative Research Center for
Advanced Computational Systems**

Talk Outline

- 1. Introduction and Motivation.**
- 2. Problems.**
- 3. A Reference Architecture.**
- 4. An Abstraction of the Reference Architecture.**
- 5. The PSI Interface.**
- 6. Results, Future Work & Conclusions.**

Introduction and Motivation

Motivation

- Convergence of *computing* and *communications* technologies.
- Unprecedented growth in demand for *networked information*.
 - Challenge of managing *persistent* data.
 - Challenge of building systems that can *scale* to meet demand.

Persistence

The management of data that outlives computations that operate over it.

- Database systems.
 - Relational.
 - Object-oriented.
 - Object-relational.
- Database programming languages (DBPLs).
 - E, Ontos etc.
- Orthogonal persistence (aka 'persistence').

Orthogonal Persistence

The most *elegant* approach to persistent data management.

Three principles:

Persistence Independence The form of a program is independent of the longevity of the data which it manipulates.

Data Type Orthogonality All data types should be allowed the full range of persistence, irrespective of their type.

Persistence Identification The choice of how to identify and provide persistent objects is orthogonal to the universe of discourse of the system.

PJama

Sun Labs, Glasgow University.

- **Two new classes:**
 - PJamaStore.
 - TransactionShell.
- **Virtual Machine modifications:**
 - Residency checks, write detection.
 - Storage backend (RVM).

Scalable Systems

Scalable hardware.

- Distributed memory computers.
- Multicomputers.
- Clusters.

Scalable software.

- Concurrency.
- Replication.
- Coherency.
- Latency.

Problems

Viewpoints

“They have been at it for 15 years and there has not been one commercial success. Orthogonal persistence is dead.”

“There is no problem. Orthogonal persistence is and should remain a research-only proposition. Commercial success is irrelevant.”

“They have been at it for 15 years and there has not been one commercial success. Orthogonal persistence is hard.”

The database and programming language communities have continued to research and develop products independently of one another, despite having to provide many similar services. [. . .]

The intellectual and software investment in each camp militates against easy adoption of the other's ideas. The dichotomy between philosophies continues and may be heightened as they view each other through caricatures.

Atkinson and Morrison 1995

However, [existing systems] do not manage to provide full database facilities—that is, few can actually demonstrate a complete repertoire of incrementality, transactions, recovery, concurrency, distribution, and scalability. (It appears that this is more a consequence of teams being unable to muster the effort to tackle all of these issues together rather than of any fundamental limits.)

Atkinson and Morrison 1995

Problems: A Summary

Commercial The lack of commercial success could be interpreted as an indication of failure of the orthogonal persistence goal.

Sociological Difficulty in bringing together disparate concerns and their disparate research groups.

Engineering Engineering task too large and broad for individual research groups to take on.

A Reference Architecture

Challenge

Construct efficient, scalable, orthogonally persistent systems.

Bigger Challenge

Develop a generalized framework for scalable persistent system design.

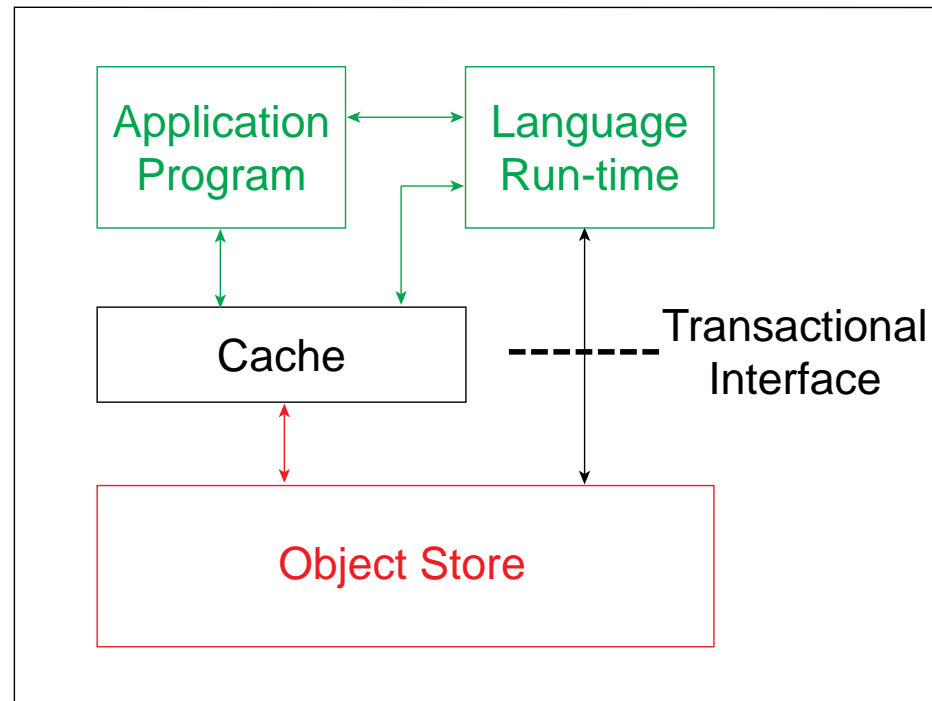
Design Factors for Scalable Persistent Systems

- **Key concerns**
 - **Concurrency**
 - **Replication**
 - **Coherency**
 - **Latency**
 - **Stability**
- **Sociological/Engineering Factors**
 - **Separation of programming language and database concerns.**
 - **Capitalization on mainstream research in respective domains.**

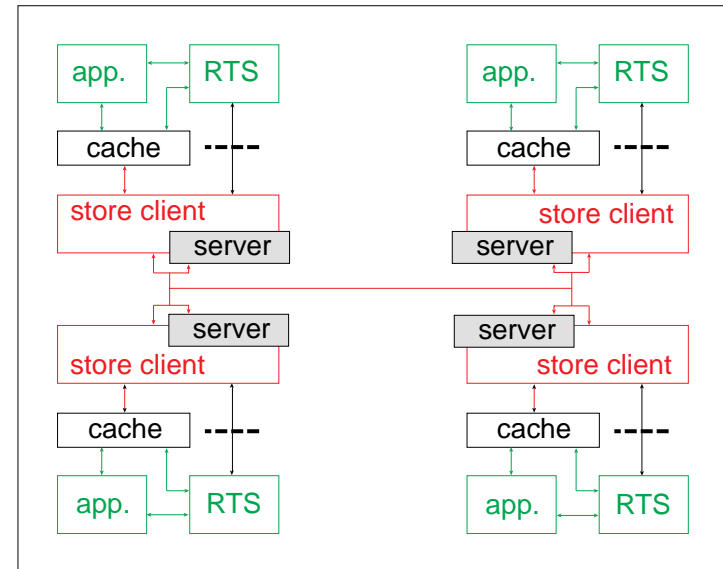
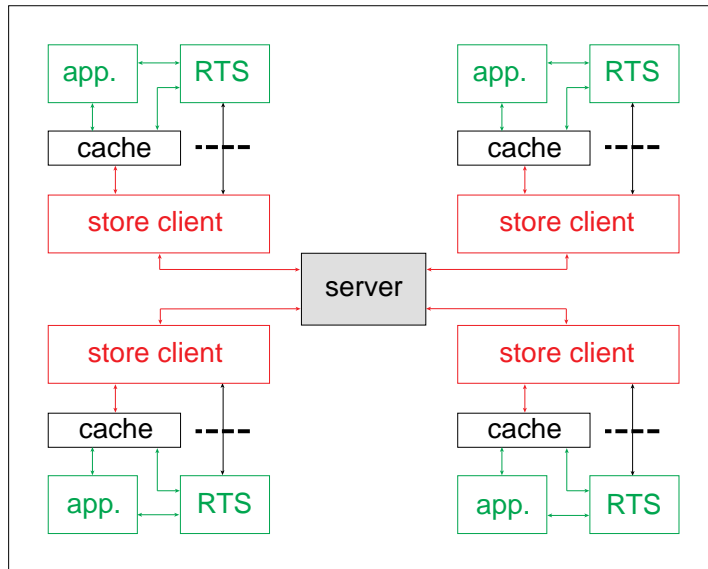
Our Weapons

- **Caching**
 - **Concurrency, Replication, Latency**
- **Atomicity (transactions)**
 - **Concurrency, Coherency, Latency, Stability**
- **Layered Architecture**
 - **Sociological and Engineering factors**

The Transactional Object Cache



Distributed Transactional Object Cache



Properties of Transactional Object Cache

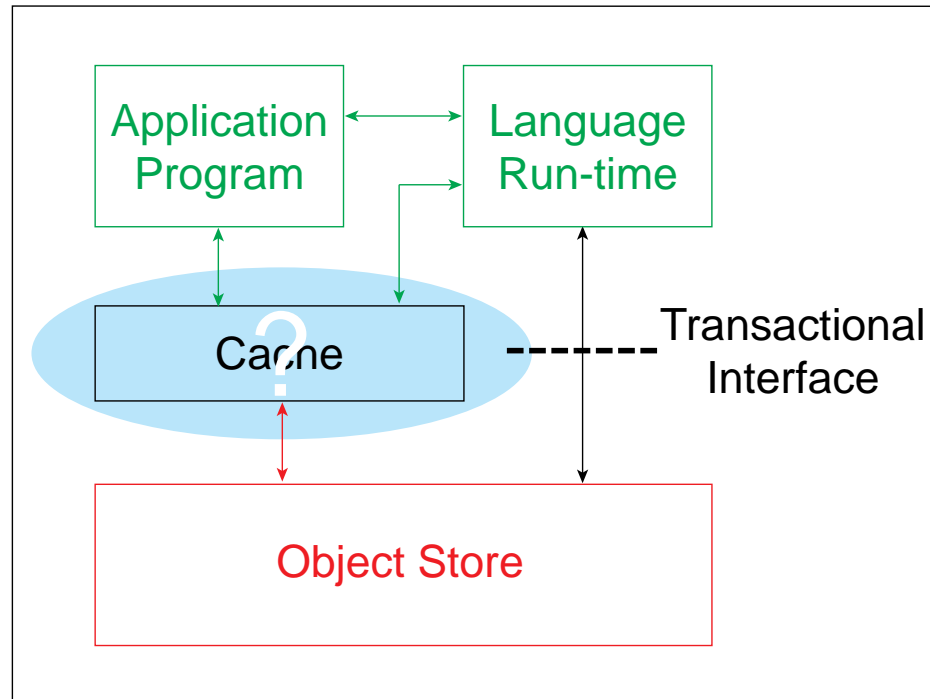
- Separation of storage and RTS concerns.
- Object-grain interface with direct memory access to store objects.
- Transactional concurrency control model.
- Distribution transparency (wrt store implementation).
- Persistence by reachability (if supporting OP).

Related Work

- ODBMS research.
 - Transactional cache coherency.
- Persistence research.
 - Mneme.

An Abstraction of the Reference Architecture

TOC Semantics



Abstraction

- **Facilitate generality.**
- **Pin down semantics.**
 - **When is my write made stable?**
 - **When may the associated buffer be freed?**
 - **When will the change be made visible to others?**
- **Separation of concerns is paramount.**
- **Identifying the right level of abstraction is vital.**

Approach

- Separation of three *orthogonal* concerns:
 - Stability.
 - Visibility.
 - Cache management.
- Develop abstractions with respect to each of these.
- Notion of ‘core’ and extended functionality.
 - Core—support for basic ACID transactions.
 - ‘Logging’—checkpoint and rollback.
 - Extended transaction models.

Stability

stability, durability and volatility

- A global *durable* stability history.
- A set of revocable local stability histories.

Core	Logging	Extended Trans.
BeginUpdates	CheckpointUpdates	DelegateUpdates
NotifyUpdate	RollbackUpdates	
AbortUpdates	StabilizeUpdates	
MakeDurable		
EvictVolatile		

Stability primitives.

BeginUpdates**NotifyUpdate****MakeDurable****BeginUpdates****NotifyUpdate****EvictVolatile****BeginUpdates****NotifyUpdate****AbortUpdates**

Stability: Core Primitives

BeginUpdates(t) $hs_t = \text{empty} \wedge HS'_l = HS_l \cup \{\langle t, hs_t \rangle\}$

NotifyUpdate(t,o) $hs'_t = hs_t.e_o$, where e_o is an event describing a change of state to some object, o .

AbortUpdates(t) $HS'_l = \{\langle t_i, hs_{t_i} \rangle \in HS_l \mid t_i \neq t\}$

MakeDurable(t)

$hs'_g = hs_g.hs_t \wedge HS'_l = \{\langle t_i, hs_{t_i} \rangle \in HS_l \mid t_i \neq t\}$

EvictVolatile $\forall \langle t, hs_t \rangle \in HS_l \quad hs'_t.s.E_c = hs_t$, where

$E_c = e_{c_0}.e_{c_1} \dots e_{c_n} \wedge s \notin E_c$

Stability: Logging Primitives

CheckpointUpdates(t) $hs'_t = hs_t.m_i$, where m_i denotes a uniquely labeled marker event.

RollbackUpdates(t,i) $hs'_t.m_i.E_r = hs_t$, where $E_r = e_{r_0}.e_{r_1} \dots e_{r_n} \wedge e_r \in \{e_u, m, s\}$, where e_u , m , and s denote update, marker and stability events respectively. In other words all events after m_i are rolled back.

StabilizeUpdates(t) $hs'_t = hs_t.s$, where s denotes a stabilize event.

Stability: Extended Transaction Models

DelegateUpdates(t_i, t_j, o)

$(hs'_{t_i} = hs_{t_i} \setminus hs^o_{t_i}) \wedge (hs'_{t_j} = hs_{t_j}.hs^o_{t_i})$, where $hs^o_{t_i}$ is a sub-history of hs_{t_i} consisting of all events e_o relating to a change in state of o , and the \setminus operator denotes history difference.

Visibility

concurrency, replication and coherency

- A single history of visibility events.
- Transactions modeled as sub-history projections.
- Multiple, possibly invalid, views of the store.
 - Both ‘avoidance’ and ‘detection’ approaches to transactional cache coherency are accommodated.
- Temporal breadth to cache interactions.
 - Read and write start and end events.

Core	Logging	Extended Trans.
<p>BeginVisibility</p> <p>ReadIntention</p> <p>ReadComplete</p> <p>WriteIntention</p> <p>WriteComplete</p> <p>AbortVisibility</p> <p>Terminated</p> <p>Finalize</p> <p>Expose</p>	<p>CheckpointVisibilty</p> <p>RollbackVisibility</p>	<p>DelegateVisibility</p> <p>IgnoreConflict</p>

Visibility primitives.

BeginVisibility
ReadIntention(c)
ReadIntention(b)
ReadIntention(d)
ReadCompletion(d)
ReadCompletion(b)
ReadCompletion(b)
Terminated
Finalize
Expose

BeginVisibility
ReadIntention(a)
ReadComplete(a)
Terminated
Finalize
AbortVisibility

BeginVisibility
ReadIntention(a)
WriteIntention(a)
WriteCompete(a)
ReadComplete(a)
Terminated
Finalize
Expose

Visibility Model

- **Attributes of object operations:**
 - **Object.**
 - **Version (partial ordering on object reads and writes).**
 - **Workspace (concurrent access to single object image).**
- **Special functions:**
 - **Sub-history termination.**
 - **Workspace isolation.**
 - **Serializability.**

Termination

$$\begin{aligned} \mathcal{T}(hv^i) = & (\forall r_o \in hv^i (\exists \bar{r}_o \in hv^i (r_o \rightarrow \bar{r}_o))) \wedge \\ & (\forall w_o \in hv^i (\exists \bar{w}_o \in hv^i (w_o \rightarrow \bar{w}_o))) \end{aligned}$$

Workspace isolation

$$\begin{aligned} \mathcal{W}(hv^i, hv^j) = & (\forall r_{ow}, \bar{r}_{ow} \in hv^i (\nexists w_{ow} \in hv^j (r_{ow} \rightarrow w_{ow} \rightarrow \bar{r}_{ow}))) \wedge \\ & (\forall w_{ow}, \bar{w}_{ow} \in hv^j (\nexists r_{ow} \in hv^i (w_{ow} \rightarrow r_{ow} \rightarrow \bar{w}_{ow}))) \end{aligned}$$

Serializability

$$\begin{aligned} \mathcal{S}(hv^i, hv^j, hv^k) = & \forall r_{ov} \in hv^i (((\exists \bar{w}_{ov} \in hv^j) \vee (\exists \bar{w}_{ov} \in (hv^i \cup hv^k))) \wedge \\ & (\nexists \bar{w}_{ov'} \in hv^j (\bar{w}_{ov} \rightarrow \bar{w}_{ov'}))) \end{aligned}$$

Visibility: Core Primitives

BeginVisibility(t) $hv^t = \text{empty} \wedge T' = T \cup \{t\}$

ReadIntention(t,o) $hv^{t'} = hv^t.r_o$

ReadComplete(t,o) $hv^{t'} = hv^t.\bar{r}_o$

WriteIntention(t,o) $hv^{t'} = hv^t.w_o$

WriteComplete(t,o) $hv^{t'} = hv^t.\bar{w}_o$

AbortVisibility(t) $(hv' = hv \setminus hv^t) \wedge (T' = T \setminus \{t\}),$

where the symbol \setminus denotes history difference and set difference respectively (i.e. the events composing sub-history hv^t are removed from hv).

Terminated(t,o) $\mathcal{T}(hv^{to})$, where hv^{to} refers to a sub-history of hv consisting of all events in transaction t relating to object o .

Finalize(t) $(\mathcal{T}(hv^t) \wedge \mathcal{S}(hv^t, hv^i, hv^{ict}) \wedge \mathcal{W}(hv^t, hv^w))$, where hv^i is the sub-history of hv consisting of all irrevocable events, hv^{ict} is the sub-history of hv consisting of all events with which t is ignoring conflicts, and $hv^w = hv \setminus (hv^t \cup hv^{ict})$.

Expose(t) $immutable(t) = true$

Visibility: Logging Primitives

CheckpointVisibility(t) $hv^{t'} = hv^t.m_i$

RollbackVisibility(t,i) $hv^{t'}.m_i.E_r = hv^t$, where

$$E_r = e_{r_0}.e_{r_1} \dots e_{r_n}$$

Visibility: Extended Transaction Models

DelegateVisibility(t_i, t_j, o)

$hv^{t_i'} = hv^{t_i} \setminus hv^{t_{i_o}} \wedge hv^{t_j'} = hv^{t_j} \cup hv^{t_{i_o}}$, where $hv^{t_{i_o}}$ is a sub-history of hv^{t_i} consisting of all events e_o relating to a change in state of o . These semantics are complex unless $\mathcal{T}(hv^{t_{i_o}})$ holds.

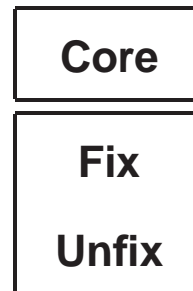
IgnoreConflict(t_i, t_j, o)

$(hv^{ic_{t_i}'} = hv^{ic_{t_i}} \cup hv^{t_{j_o}}) \wedge (hv^{ic_{t_j}'} = hv^{ic_{t_j}} \cup hv^{t_{i_o}})$.

Thus for all events relating to object o , t_i and t_j are added to each other's 'ignore conflict' sub-histories (hv^{ic_t}). A subsequent call to Finalize will thus ignore conflicts between h_{t_i} and h_{t_j} for those events.

Cache Management

availability



The PSI Interface

Defining interfaces is the most important part of system design. Usually, it is also the most difficult, since the interface must satisfy conflicting requirements:

- **An interface should be simple.**
- **It should be complete.**
- **It should admit a sufficiently small and fast implementation.**

Butler W. Lampson

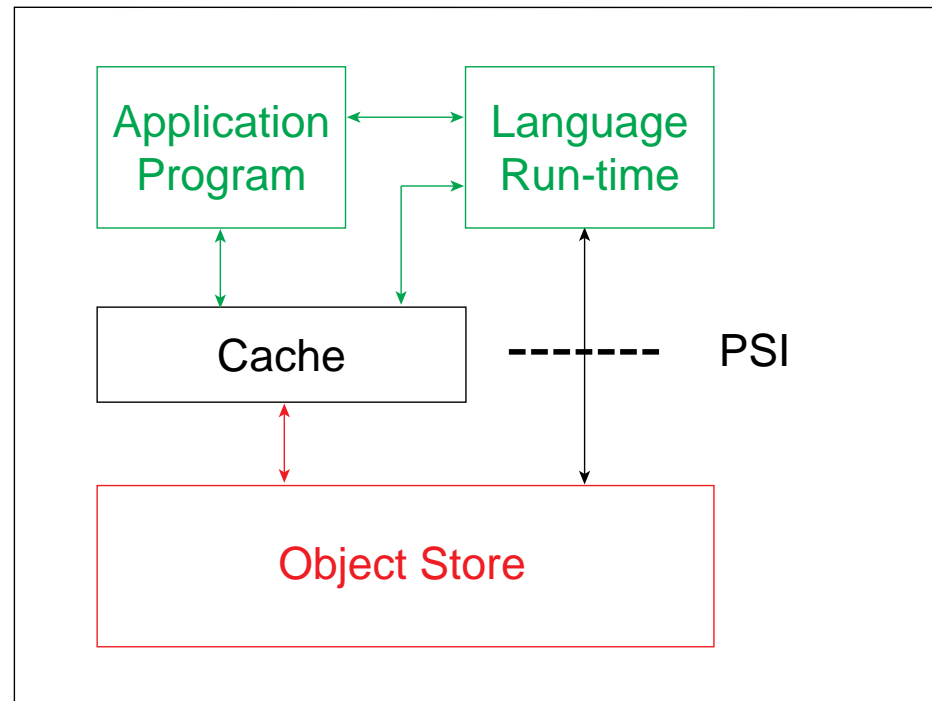
Contrasting Approaches

- MPI (Message Passing Interface).
- RVM (Recoverable Virtual Memory).

Key PSI Design Goals

- Flexibly support the needs of PPLs.
- Strongly separate concerns.
- Admit efficient implementations.

PSI and the Transactional Object Cache



Separation of Concerns

- **Persistence Identification.**
 - PBR from a single root, GC below interface.
- **Residency checks and write detection.**
 - RTS dependent.
- **Swizzling.**
 - RTS dependent, hooks for long OID support.
- **Concurrency control and cache management.**
 - Mechanisms hidden, RTS given levers.

- **Recovery.**
 - **Mechanisms hidden, RTS given levers.**
- **Advanced storage structures.**
 - **Optimized index facilities.**

The PSI Core Interface

**PSI_Read, PSI_Write, PSI_New, PSI_NewTrans, PSI_Commit,
PSI_Abort, PSI_Unfix**

- **PSI_Read**
 - **ReadIntention and Fix semantics.**
 - **Copy-out allowed (no Fix).**
- **PSI_Write**
 - **WriteIntention and Fix semantics.**
 - **Target object may be in-place or private.**

- **PSI_New**
 - **WriteIntention and Fix.**
 - **Creates new object, takes placement hint, instance associated with a “descriptor”.**
- **PSI_NewTrans**
 - **Returns a transaction handle.**
- **PSI_Commit**
 - **Unfix, ReadComplete, WriteComplete, and NotifyUpdate; conditional Finalize, MakeDurable and Expose.**

- **PSI_Abort**
 - **AbortUpdates, AbortVisibilty and Unfix.**
- **PSI_Unfix**
 - **Unfix, conditional NotifyUpdate.**

PSI_NewTrans

PSI_Read(c)

PSI_Read(b)

PSI_Read(d)

PSI_Commit

PSI_NewTrans

PSI_Read(a)

PSI_Commit

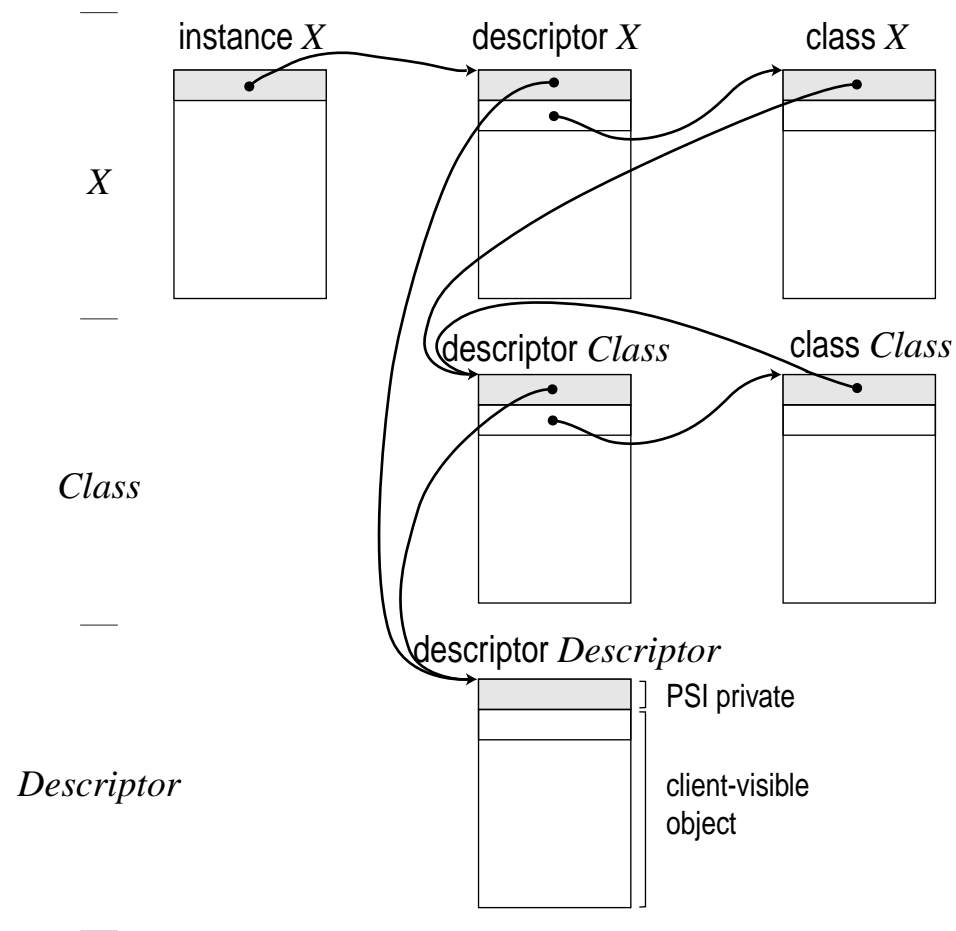
PSI_NewTrans

PSI_Read(a)

PSI_Write(a)

PSI_Commit

PSI Descriptors

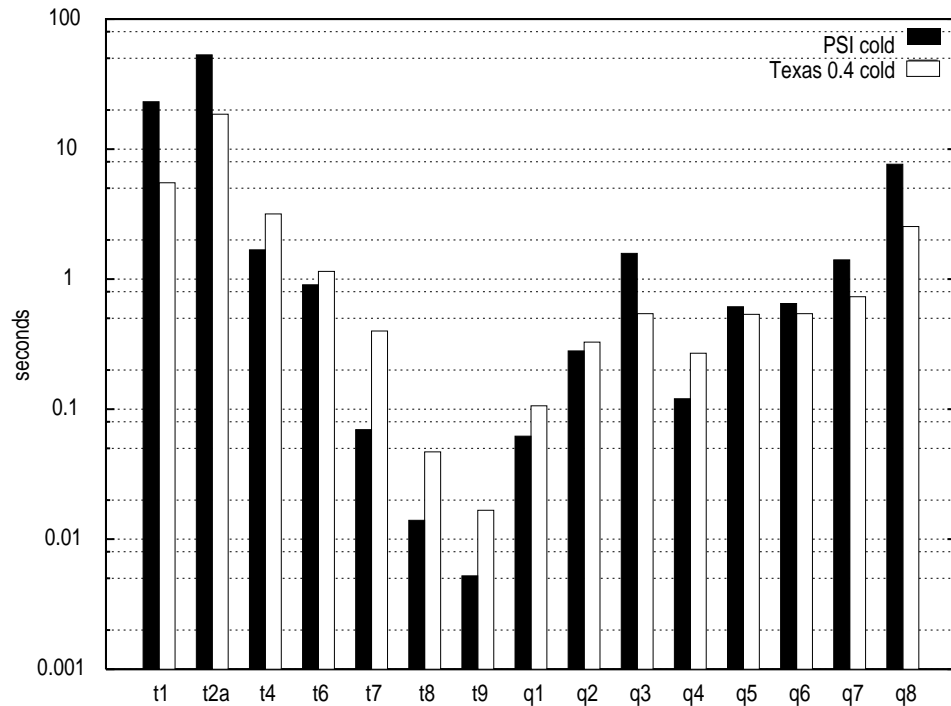
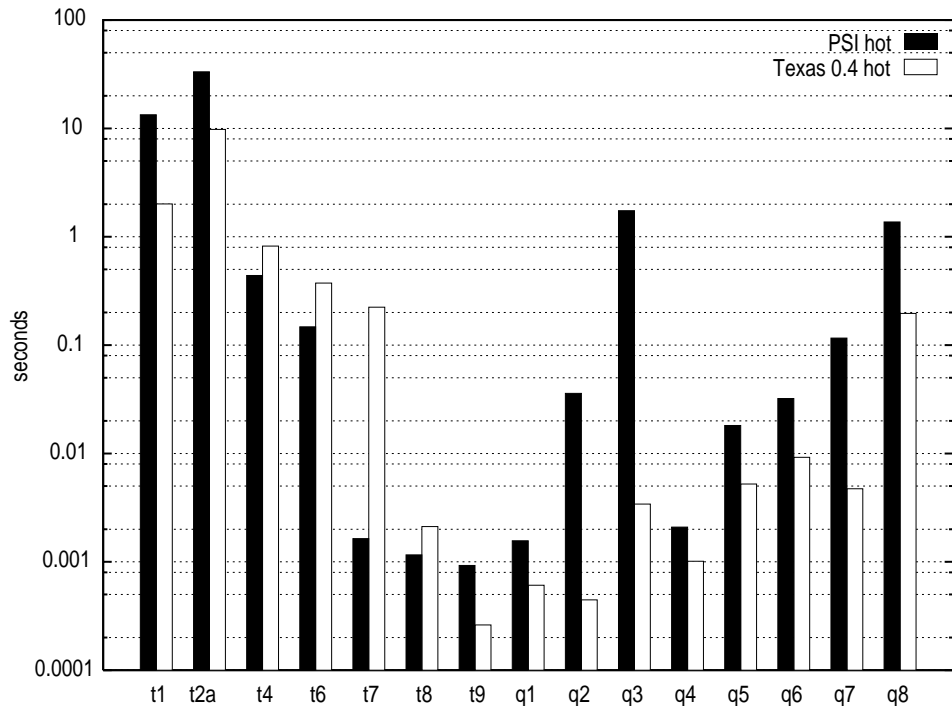


Results, Future Work & Conclusions

Results

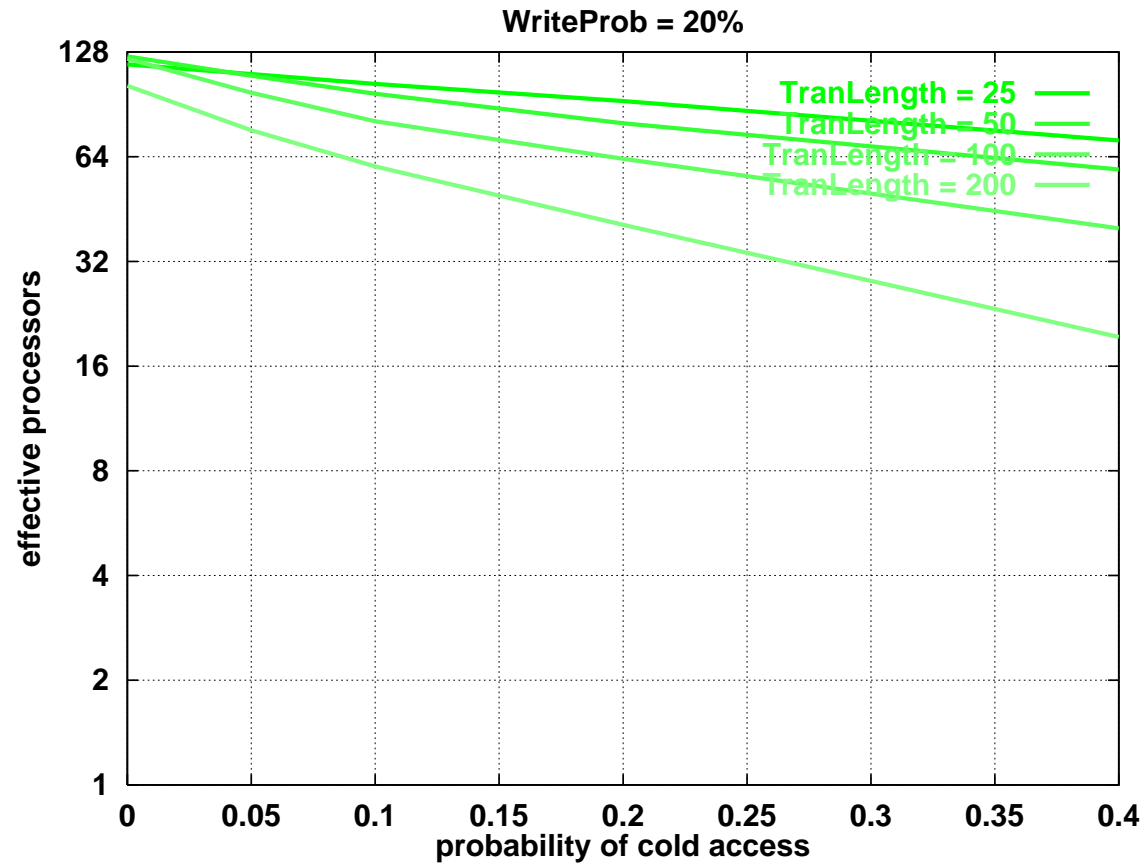
- **Single-user PSI implementation built.**
 - **Proof of concept.**
 - **No lock manager and no GC.**
- **Multicomputer PSI implementations.**
 - **AOCC concurrency control experiments.**
 - **Highly scalable concurrency control.**
 - **Server scalability experiments.**

OO7 Benchmark



'small' database traversal times

PSI/AOCC Scalability



HOTCOLD workload on 128 node Fujitsu AP1000

Development Plans

- **PSI/Persistent Java.**
- **Publicly available single-user PSI implementation.**
- **Avoidance based cache coherency (e.g. PS/AA).**
- **Distributed GC.**
- **Multicomputer Persistent Java on MC PSI implementation.**

Application Development

- Toy applications.
- Very large business model (ABS).
- ???

Recap

- Scalable information servers.
- Elegance of orthogonal persistence.
- Difficulty of orthogonal persistence.
- Reference architecture captures key concerns through *caching, atomicity* and *layering*.
- PSI—instantiation of a general OP construction framework.
- PSI facilitates focusing of research and adoption of mainstream technologies.

Conclusions

- **A generalized framework for (orthogonally) persistent system construction has been identified.**
- **The framework has been realized in the form of PSI.**
- **This development gives cause for optimism that orthogonal persistence will play an important role in the future of scalable information management.**

The End