Distributed Pool and its Implementation

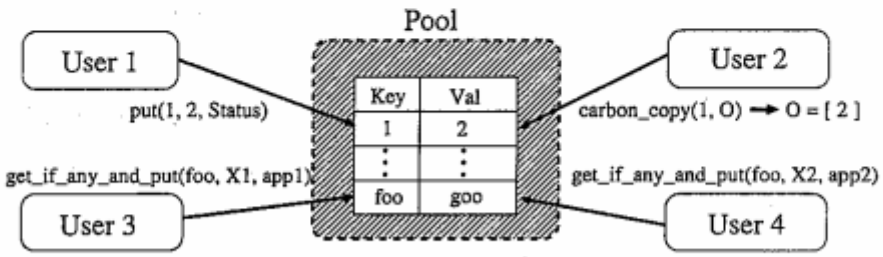
Masaki Sato

Institute for New Generation Computer Technology

Contents

What is a "Pool" ?

- Basic in-memory database features such as hash tables
- Used extensively by most applications
- Reduced programming efforts



Problems in a Pool

- A pool is reprecented as a single process
- Application processing large amounts of data, and managing all the data as a unit on a distributed environment
  - Longer latency for data accesses
  - Concentration of computing and communication load
  - Data concentration at one particular node

→ **Distributed pool using software caching**

Features of our Cache Protocol (1)

- **Asynchronous communication**

  A state transition scheme containing temporary states defined between sending a message and receiving a reply

- **Nonexistent back storage**

  A distributed cache on each node consistutes whole storage. An owner cache is responsible for supplying data and continues to keep it.

Features of our Cache Protocol (2)

- **Many message interfaces for the user**

  Our cache coherent mechanism is further complicated

- **Search keys arbitrarily chosen by the user**

  No access localities associated with similarities of keys

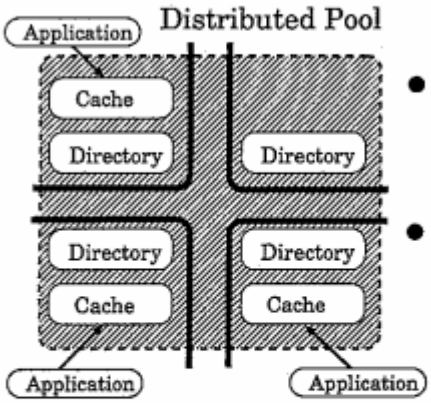
- **Write-invalidate type cache consistency**

  Decreases the amount of data sharing

- **Replacement**

  Managing data except owned data with the FIFO algorithm

Components of Distributed Pool



- Cache processes

  keep data (pointer) in a hash table

- Directory processes

  manage cached data locations

  transfer messages between caches

Cache States (1)

Permanent states

Invalid(I) : the cache does not contain this data.

Exclusive(E) : no other cache contains this data.

Shared-Owned(SO) : other caches possibly contain this data, but this cache is responsible for suppling the data and continues to keep it.

Shared-Unowned(SU) : at least one other cache contains this data and this cache does not take any responsibility for keeping it.

Cache States (2)

Temporary states

States	Waiting for exclusive access to data	Keeping data	Next state (permanent)
WE	○	○	E
WEI	○	○	E or I
WSD	×	×	SO or I
WP	○	○	I
WED	○	×	E
WEID	○	×	E or I
WPD	○	×	I

Directory States

Permanent state

Invalid(I) : no cache contains this data.

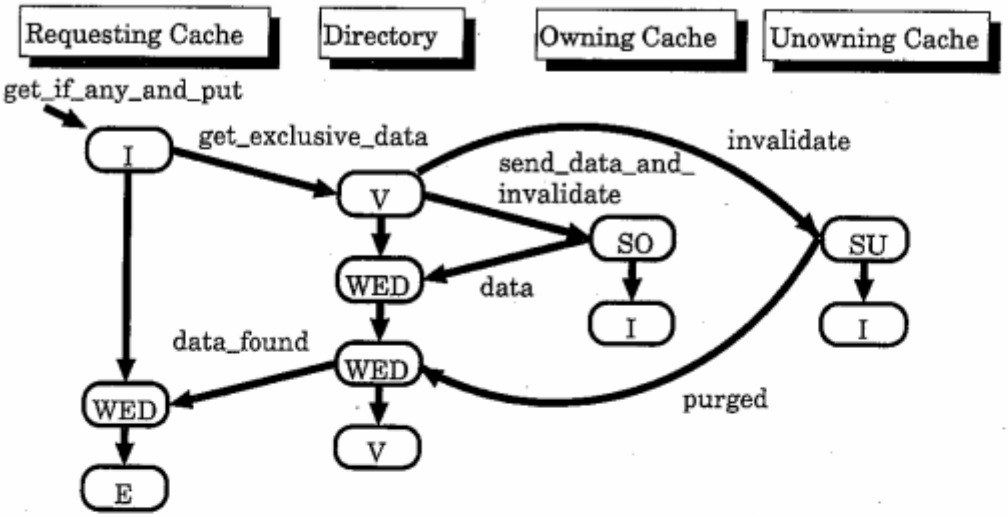
Valid(V) : some caches contain this data.

Temporary states

States	Waiting for exclusive access to data	Waiting for data	Next state (permanent)
WD	×	○	V or I
WED	○	○	V
WEDI	○	○	V or I
WE	○	×	V
WRD	○	○	I
WP	○	×	I

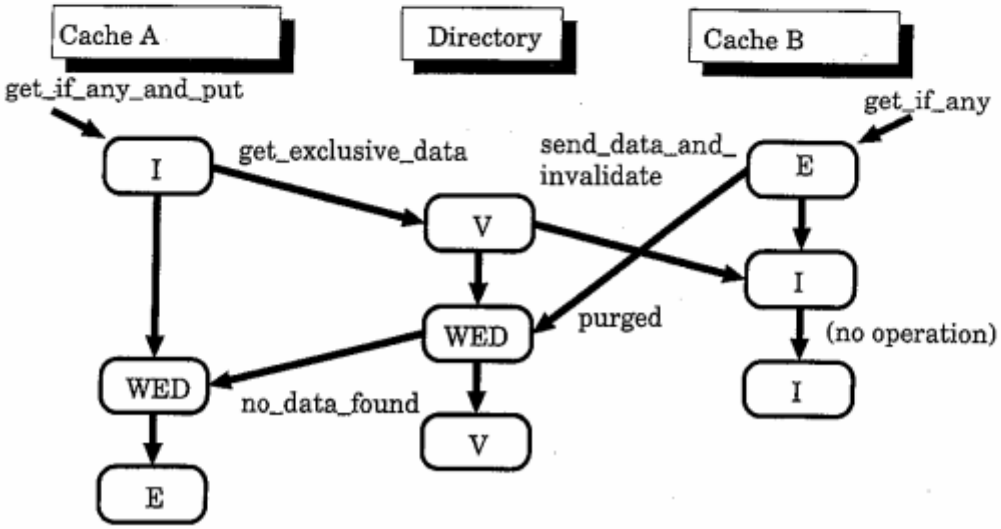
Coherence Protocol (1)

(Ex.1) Receiving a Copy-and-Update Request in an Invalid State



Coherence Protocol (2)

(Ex.2) Crossed Messages



Evaluation

Comparison with a conventional centralized pool

- Reduction of interprocessor communication by caching
- Distribution of computing load

Hardware

- PIM/m
- Sparc Center 2000

  PVM version of the distributed KLIC system

  - Lazy transfer mode
  - Eager transfer mode

Reduction of interprocessor communication by caching

Measurement of response time of carbon_copy messages

- Integer data
- List data containing ten integer elements

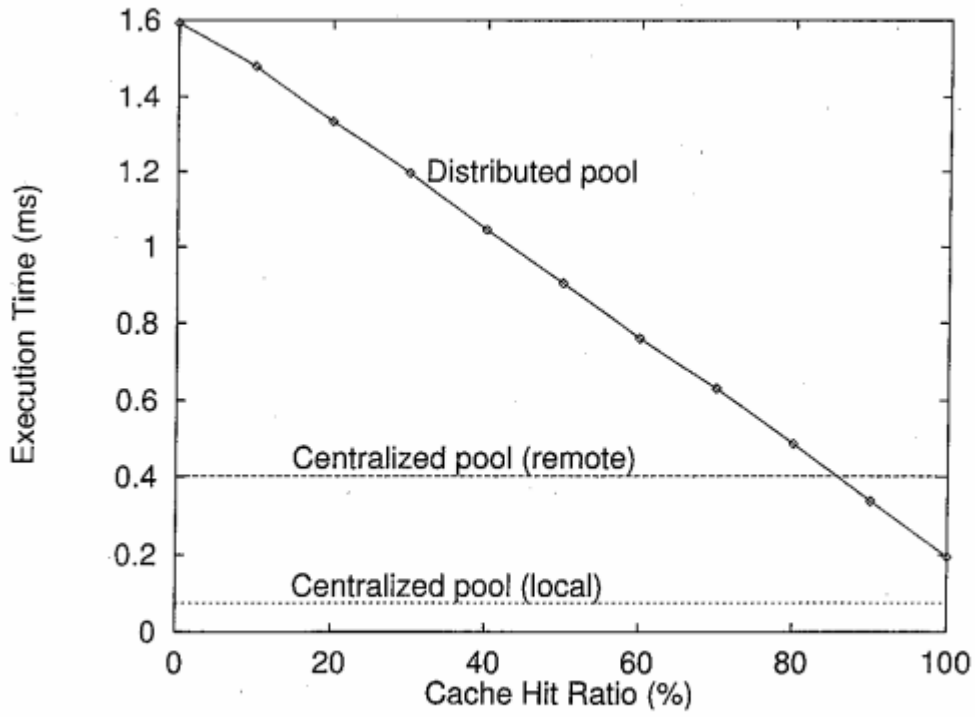
Centralized pool

- Location of a pool process and a user process
  - on same node → local access
  - on different nodes → remote access

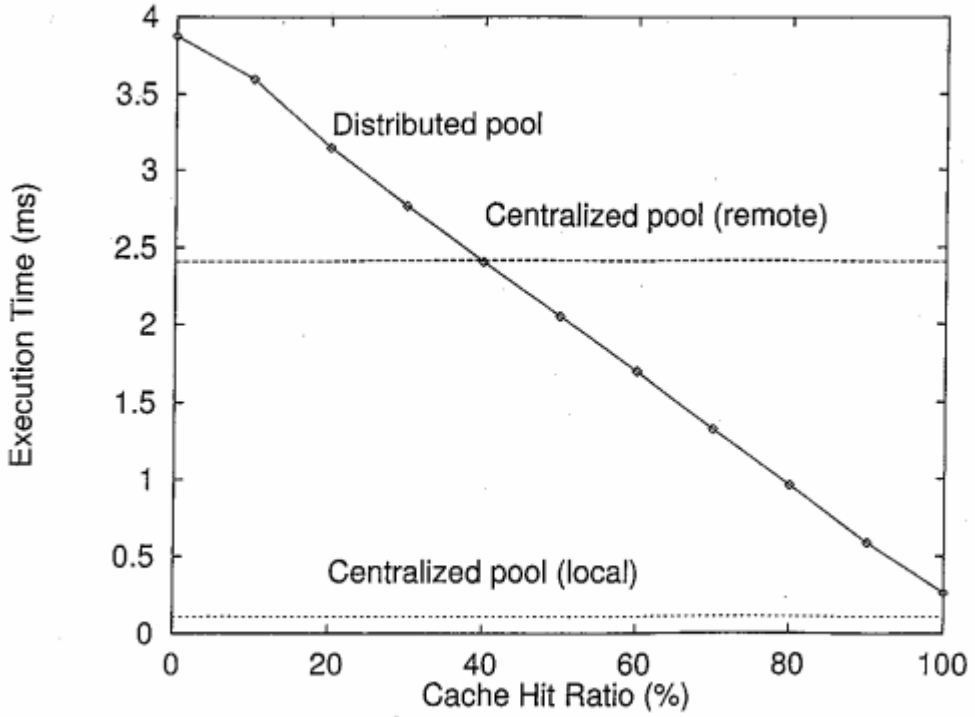
Parameter

- Hit ratio for a distributed pool

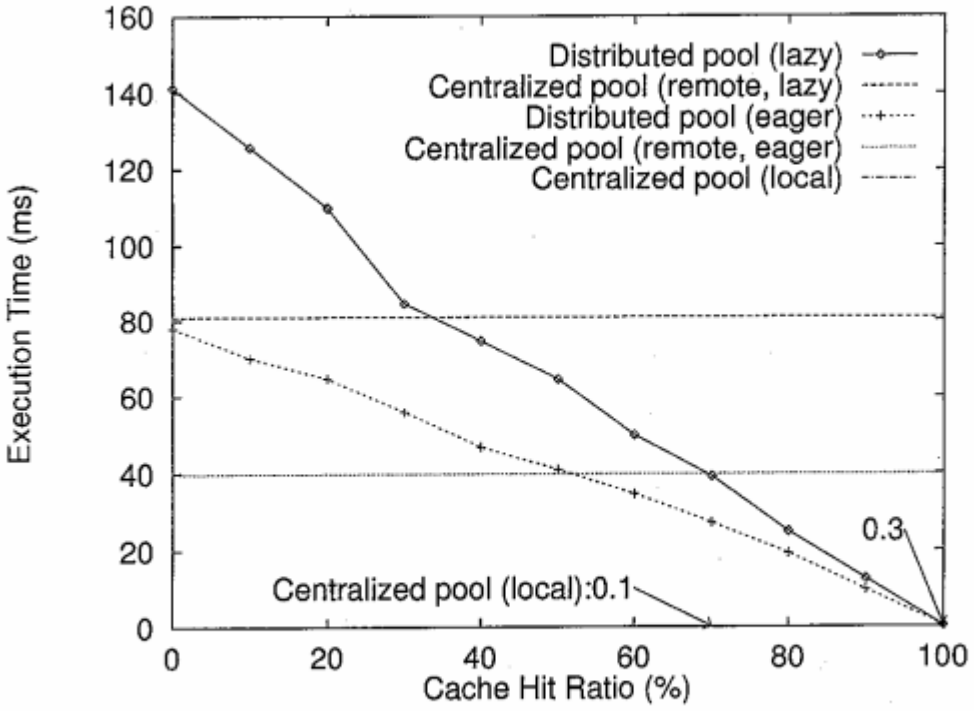
Execution Time on PIM/m (Integer)



Execution Time on PIM/m (List)



Execution Time on KLIC (Integer)



Distribution of Computing Load

Measurement of the execution time for copying 1000 integers of data

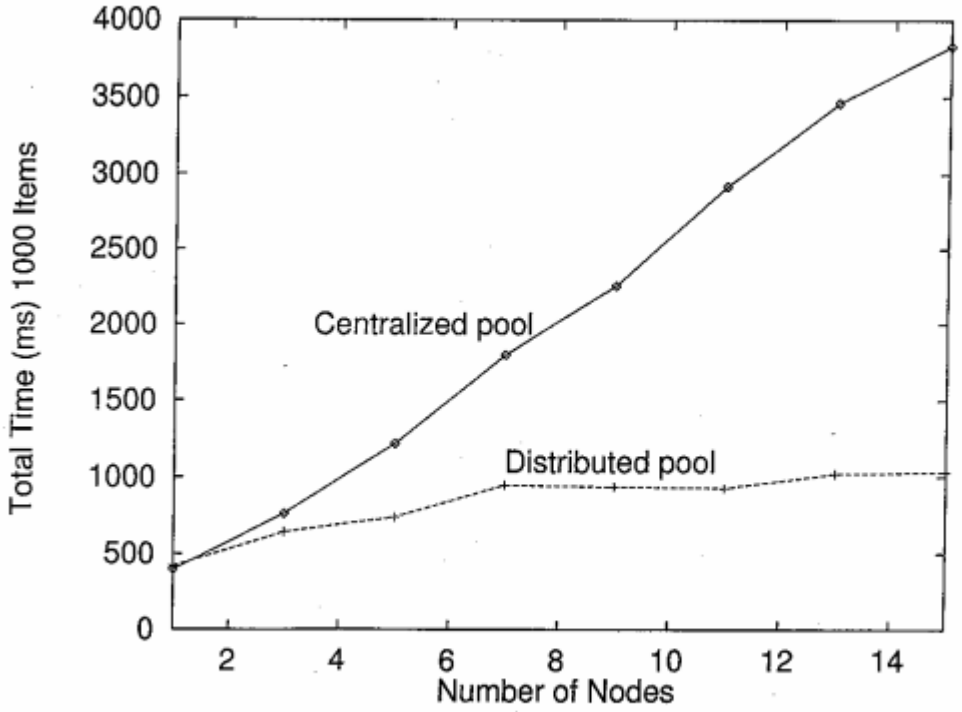
Centralized pool

- Pool process : 1
- User processes : 1 ~ 15

Distributed pool

- Cache and directory processes : 1 ~ 15
- User processes : 1 ~ 15
- Hit ratio : 85%

Execution Time for Concentrating Access on PIM/m



Conclusion

We have introduced the distributed pool, which distributes data efficiently among many processing nodes.

- Interprocessor communication was reduced.
- Access concentration was eased.

Using the distributed pool, application programmers

- can distribute the computing load flexible without worrying about data consistency.
- uses memory more efficiently for applications processing large amounts of data.