Efficient Global Object Space Support for Distributed JVM on Cluster

Weijian Fang, Cho-Li Wang and Francis C. M. Lau
The Systems Research Group
Department of Computer Science and Information Systems
The University of Hong Kong

Outline

- Introduction
  - Distributed Java Virtual Machine
  - Global Object Space
  - Related Works
- Our Approach
  - Cache Coherence Protocol
  - Distributed-Shared Object
  - Optimizations
- Performance Evaluation
- Conclusion and Future Work

Motivation #1 : Java

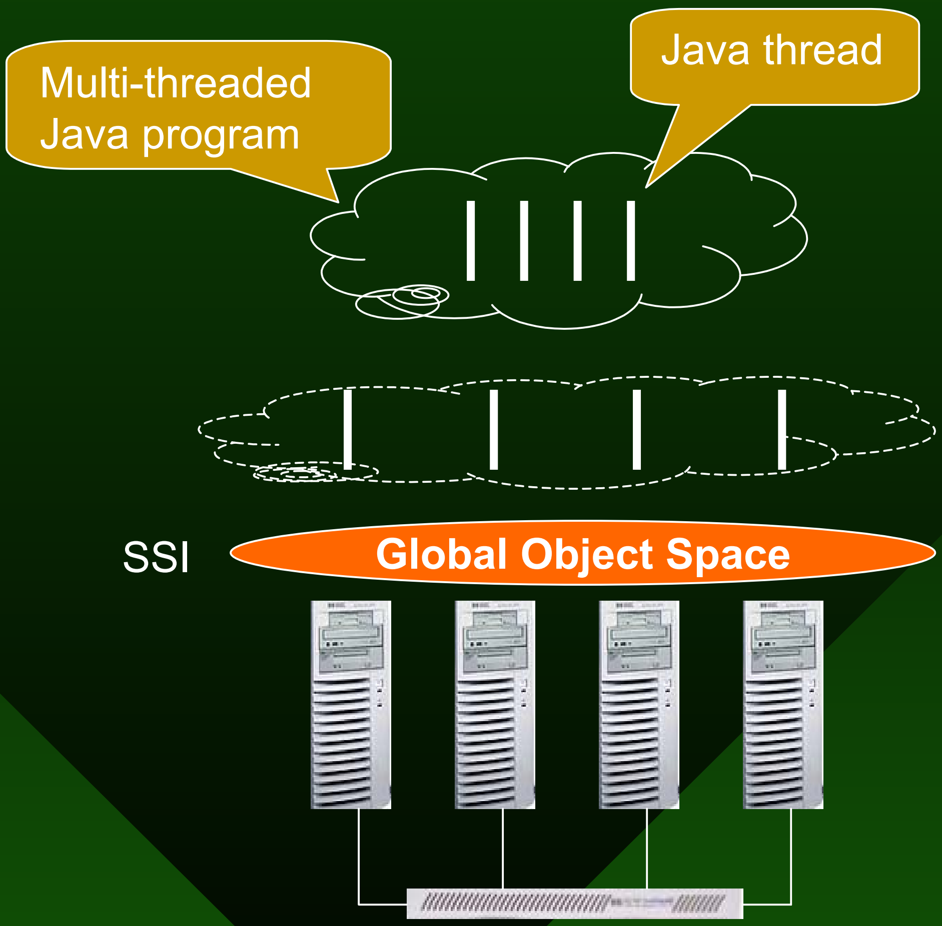
- Built-in multi-threading
  - A parallel programming language?
- High performance
  - "Java has potential to be a better environment for Grande application development than any previous languages such as Fortran and C++. "
    - – Java Grande Forum. http://www.javagrande.org/.

Motivation #2 : Cluster Computing

- Cost effective parallel computing
  - Open source software
  - Commodity hardware
- Until June 2002 (www.top500.org)
  - 80 of top 500 supercomputers are clusters
  - The 3rd powerful supercomputer in the world is a cluster
    - 750 HP AlphaServer ES45 connected by Quadrics interconnection network

Distributed JVM

- Comply with JVM Spec.
  - Transparent execution of multi-threaded Java programs
- Present a *Single System Image* of cluster to Java programs
  - Automatic distribution of Java threads among cluster nodes



Global Object Space Support

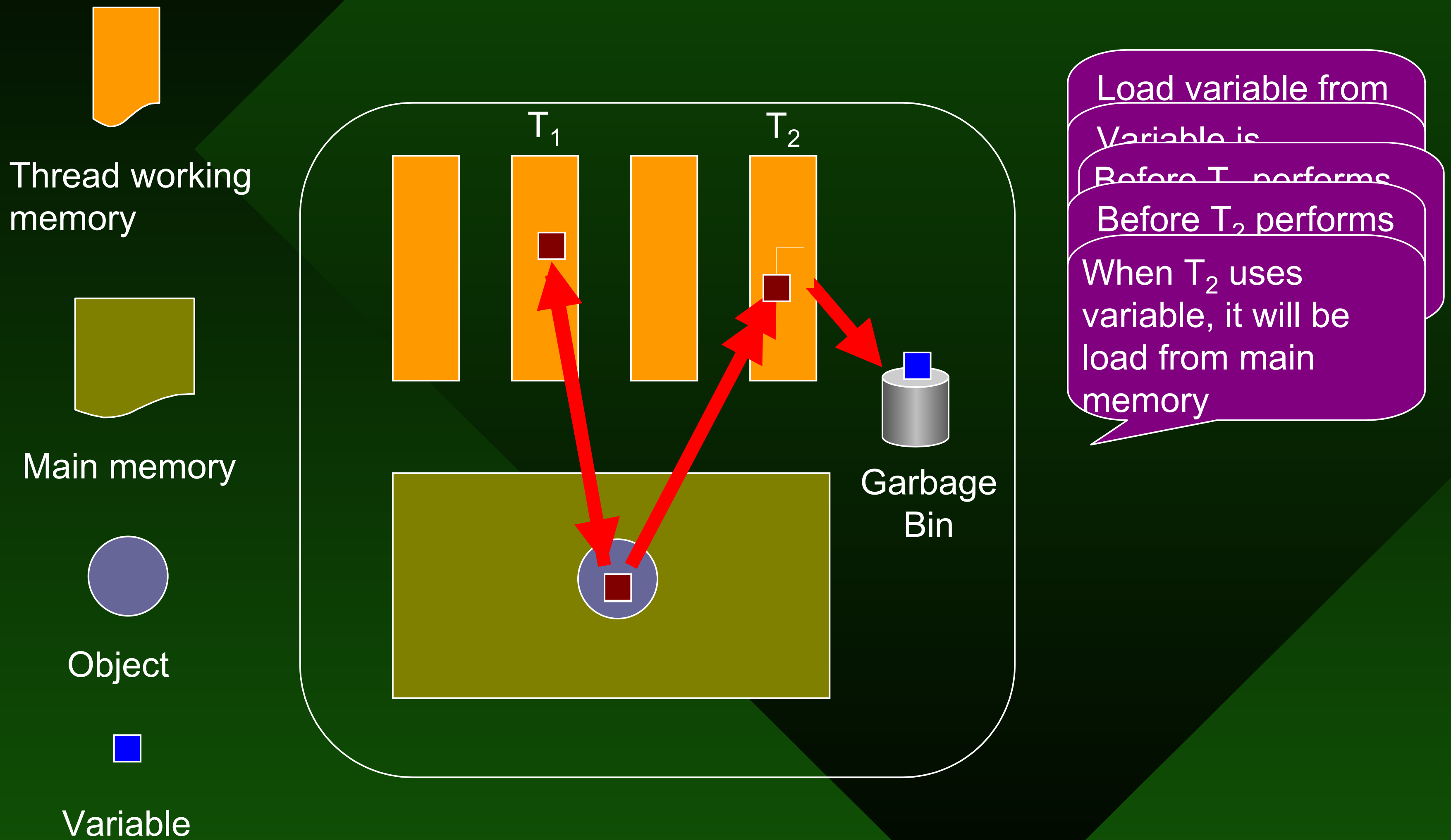
- Transparency
  - Transparent object access disregarding thread/object's physical location
    - Virtualize a single object heap spanning on the whole cluster
  - A distributed shared memory service
- Consistency
  - Comply with Java Memory Model to handle the memory consistency issue
- Efficiency
  - Reduce the network traffic incurred by distributed computing of Java threads

Our goal is to design and implement an efficient GOS for distributed JVM.

Java Memory Model

- Define memory consistency semantics in multi-threaded Java programs
  - GOS must comply with JMM
- There is a lock associated with each object
  - Protect critical sections
  - Maintain memory consistency between threads
- JMM is similar to Home-based Lazy Release Consistency

Java Memory Model (contd.)



Related Works

- Method shipping
  - Usually no replication
  - Method invocation and object access will be forwarded to the node where object resides
  - E.g. cJVM
- Page shipping
  - Leverage page-based DSM to build GOS at runtime
  - E.g. JESSICA, Java/DSM
- Object shipping
  - Leverage object-based DSM to build GOS at runtime
  - E.g. Hyperion, Jackal

Method Shipping

- E.g. cJVM
- Master/proxy object model
  - Method invocation and field access on proxy object should be forwarded to master object.
- Usually forbid object replication to leave out consistency problem
- More aggressive object caching is preferred
- Load distribution is determined by object distribution

Page Shipping

- E.g. JESSICA, Java/DSM
- Leverage some page-based Distributed Shared Memory
- Sharing granularity gap between object-oriented Java and page-based DSM
  - False sharing problem
- Not easy to do further optimization

Object Shipping

- Leverage some object-based DSM at run time
- **Examples:**
  - **Hyperion**: translate Java bytecode to C
  - **Jackal**: compile Java source code directly to native code
- No JVM involved in execution

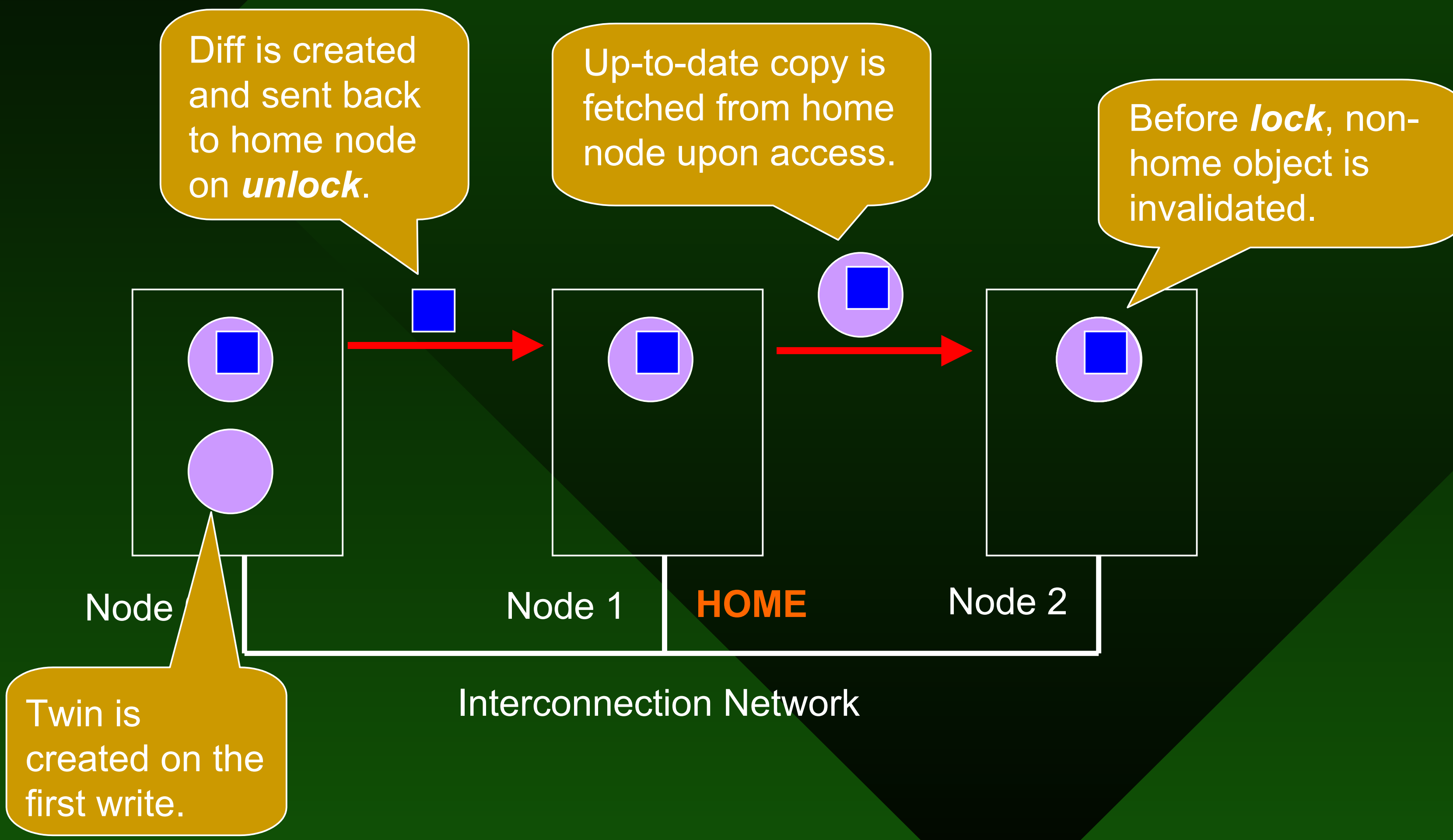
Outline

- Introduction
  - Distributed Java Virtual Machine
  - Global Object Space
  - Related Works
- Our Approach
  - Cache Coherence Protocol
  - Distributed-Shared Object
  - Optimizations
- Performance Evaluation
- Conclusion and Future Work

A Straight-forward Object-based Cache Coherence Protocol for JMM

- Home-based
  - A home node is selected for each object
  - Updates are propagated to the home on synchronizations
  - Clean copies are derived from the home
  - Home node acts as lock manager
- Twin and Diff
  - Support concurrent multiple writer

Example


Diff is created and sent back to home node on ***unlock***.
Up-to-date copy is fetched from home node upon access.
Before ***lock***, non-home object is invalidated.
Node
Node 1
HOME
Node 2
Twin is created on the first write.
Interconnection Network

DSO - Definition

- Object connectivity and thread reachability are available at run time
- Consider reachability
  - ***Thread-local object***: reachable from only one thread
  - ***Thread-escaping object***: reachable from multiple threads
- Further consider the physical locations of thread and object in distributed JVM
  - ***Node-local object (NLO)***: reachable from thread(s) at the same node
  - ***Distributed-shared object (DSO)***: reachable from at least two threads located at different cluster nodes

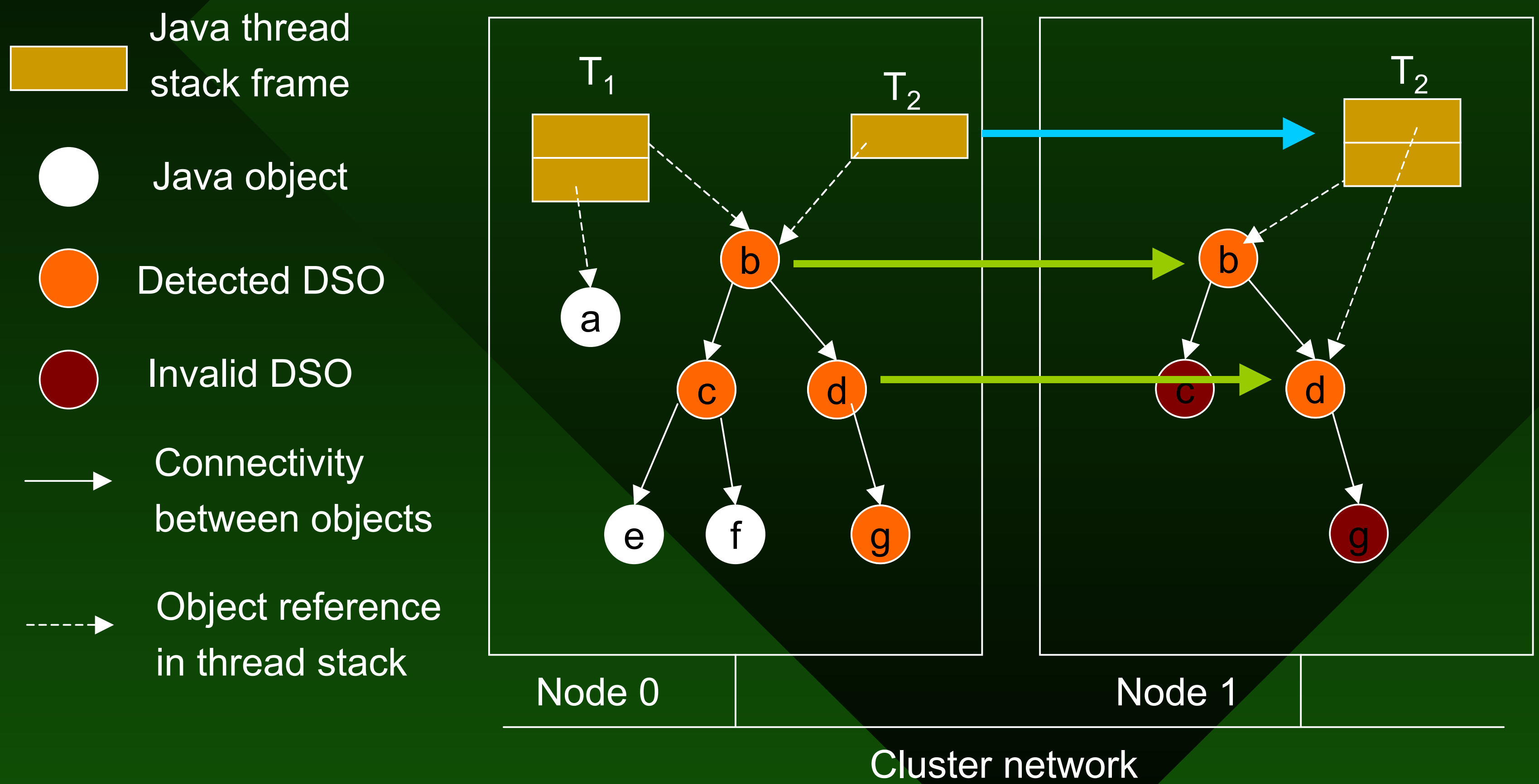
DSO – Benefits from DSO detection

- Only synchronizations on DSOs should trigger distributed consistency operation
- Only DSOs are involved in distributed consistency operation
- NLOs can be safely collected by a local garbage collector

DSO – A lightweight detection scheme

- Leverage Java's runtime reachability information
- The detection is postponed upon
  - The distribution of Java threads to other nodes
  - Sending objects to a remote node
- Identify object references transmitted to other nodes
  - Must be DSOs

DSO – Detection (Ex.)



Optimizations

- Object Home Migration
- Synchronized Method Migration
- Object Pushing

Object Home Migration

- Access asymmetry in home-based protocol
  - Coherence of home copy is kept through update
  - Coherence of non-home copy is kept through invalidate
  - Home accesses are more lightweight than non-home accesses
- Home migration
  - Reselect the node where most accesses happen as the home node for the object
  - Adapt to object access behavior in applications
  - Negative impact
    - Migration notices

Object Home Migration (contd.)

- Optimize object exhibiting single writer access pattern
- Record remote writes at home node
  - Remote writes come as diff messages
- Count consecutive writes
  - Issued by the same remote node
  - Not interleaved by writes from other nodes
- Migrate home to the writing node
  - When the number of consecutive writes exceeds a predefined threshold

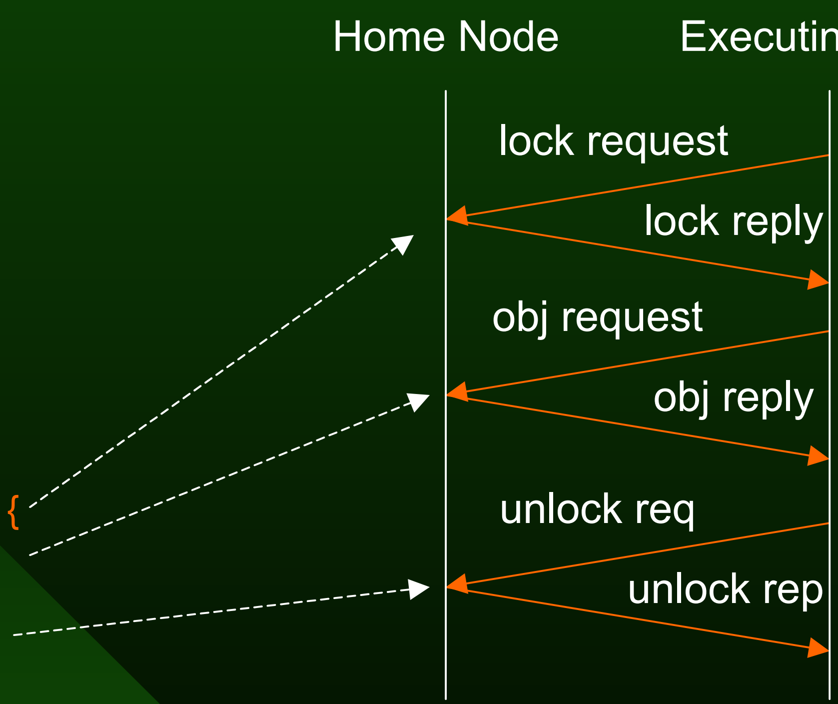
Synchronized Method Migration

inc() is invoked on a non-home node

```
class Counter {
    private int i; // internal counter

    public Counter() {
        i = 0;
    }

    public synchronized void inc() {
        i++;
    }
}
```



Non-home execution of synchronized method involves multiple message roundtrips

Synchronized Method Migration (contd.)

- Non-home execution of synchronized method is usually inefficient in distributed JVM
  - Involves multiple message roundtrips
- Migrate synchronized method of DSO to its home node for execution
  - Only one message roundtrip
  - Aggregate synchronization and data requests
- Thanks to the detection of DSOs

Object Pushing

- Reference locality
  - After an object is accessed, its reachable objects are very likely to be accessed afterwards.
  - Partially determined by reachability
  - Prefetching
- Object pushing
  - Push-based prefetching
  - The home node pushes the objects reachable from the requested DSO
  - Reachability information at home node is always valid
    - Guarantee the correctness of prefetching
- Optimal message length
  - Represent preferred aggregation size of objects

Outline

- Introduction
  - Distributed Java Virtual Machine
  - Global Object Space
  - Related Works
- Our Approach
  - Cache Coherence Protocol
  - Distributed-Shared Object
  - Optimizations
- Performance Evaluation
- Conclusion and future work

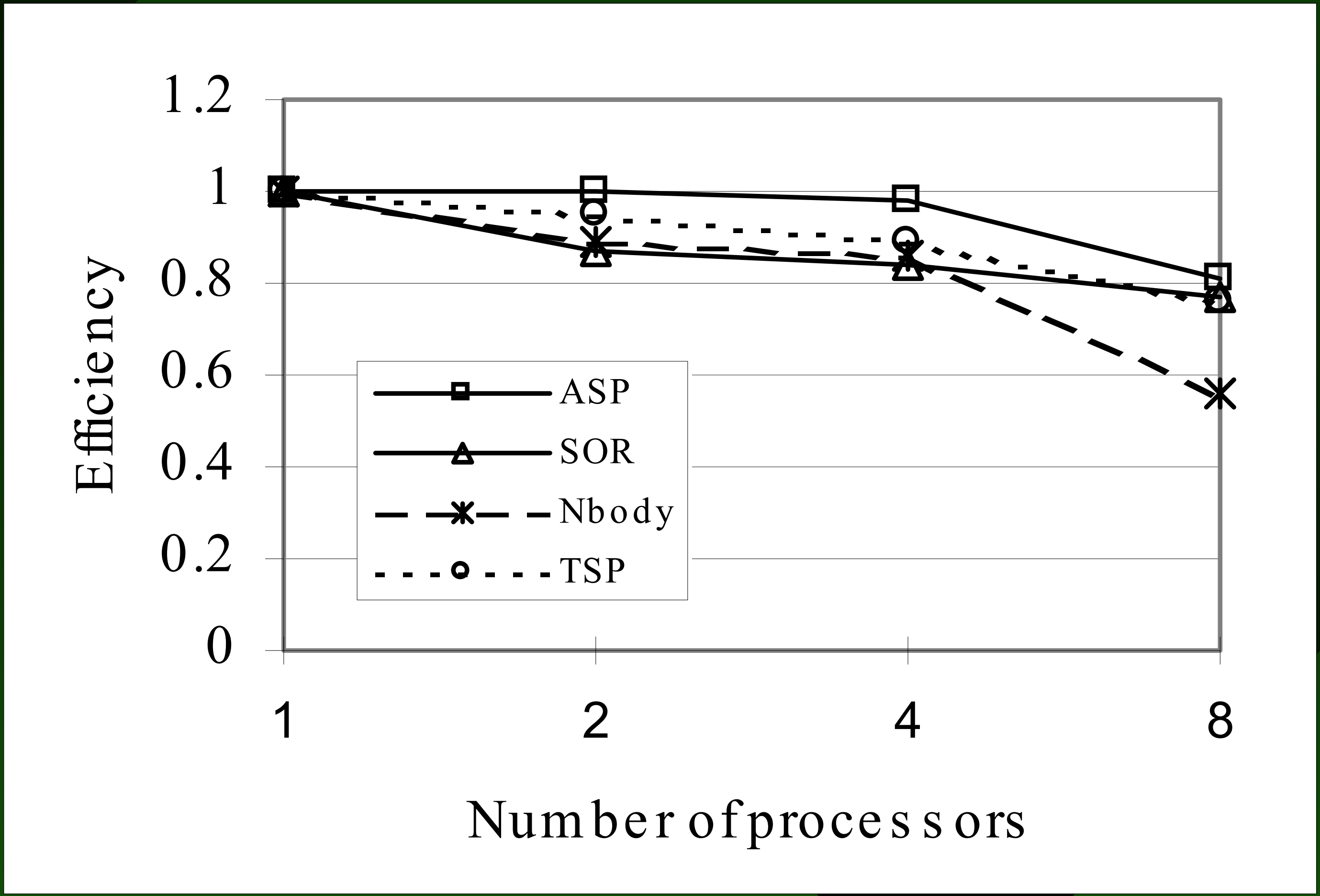
Implementation

- Modify Kaffe 1.0.6
- On a cluster of 300MHz PII PCs, running Linux 2.2, connected by Fast Ethernet
- Threads are automatically distributed among cluster nodes

Benchmark Suite

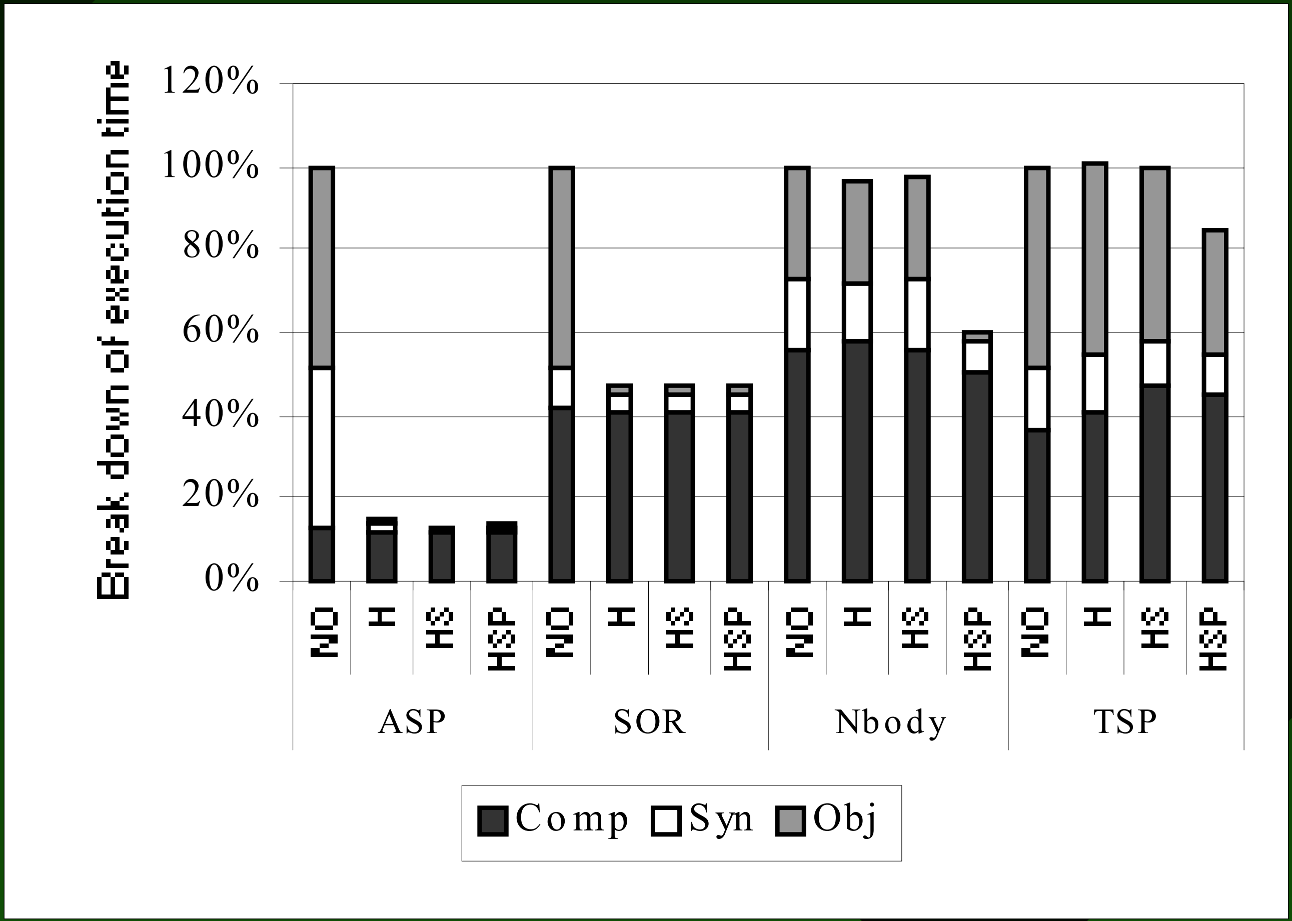
- ASP (All-pair Shortest Path)
- SOR (red-black successive over-relaxation)
- Nbody
- TSP

Efficiency

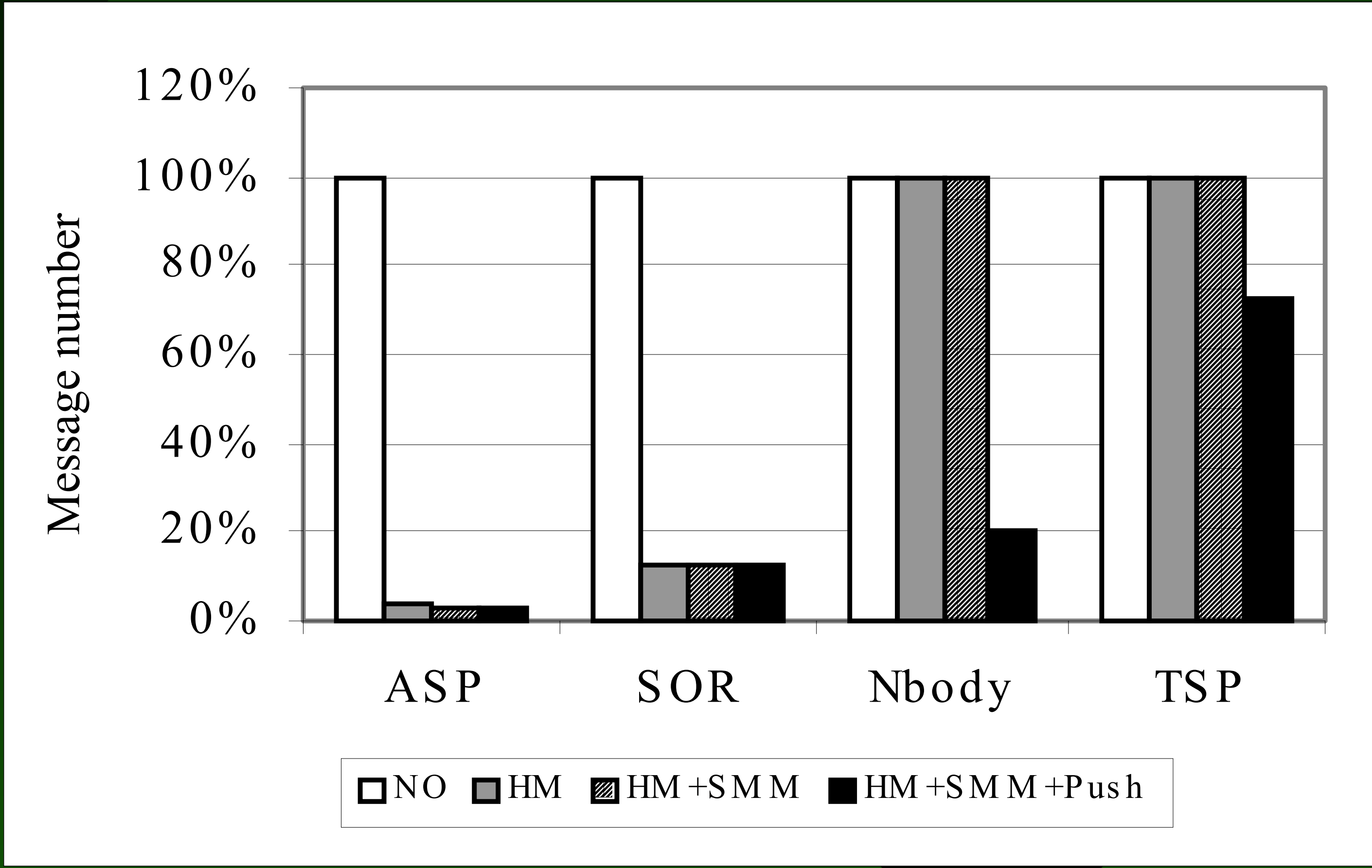


Efficiency
1.2
1
0.8
0.6
0.4
0.2
0
ASP
SOR
Nbody
TSP
1
2
4
8
Number of processors

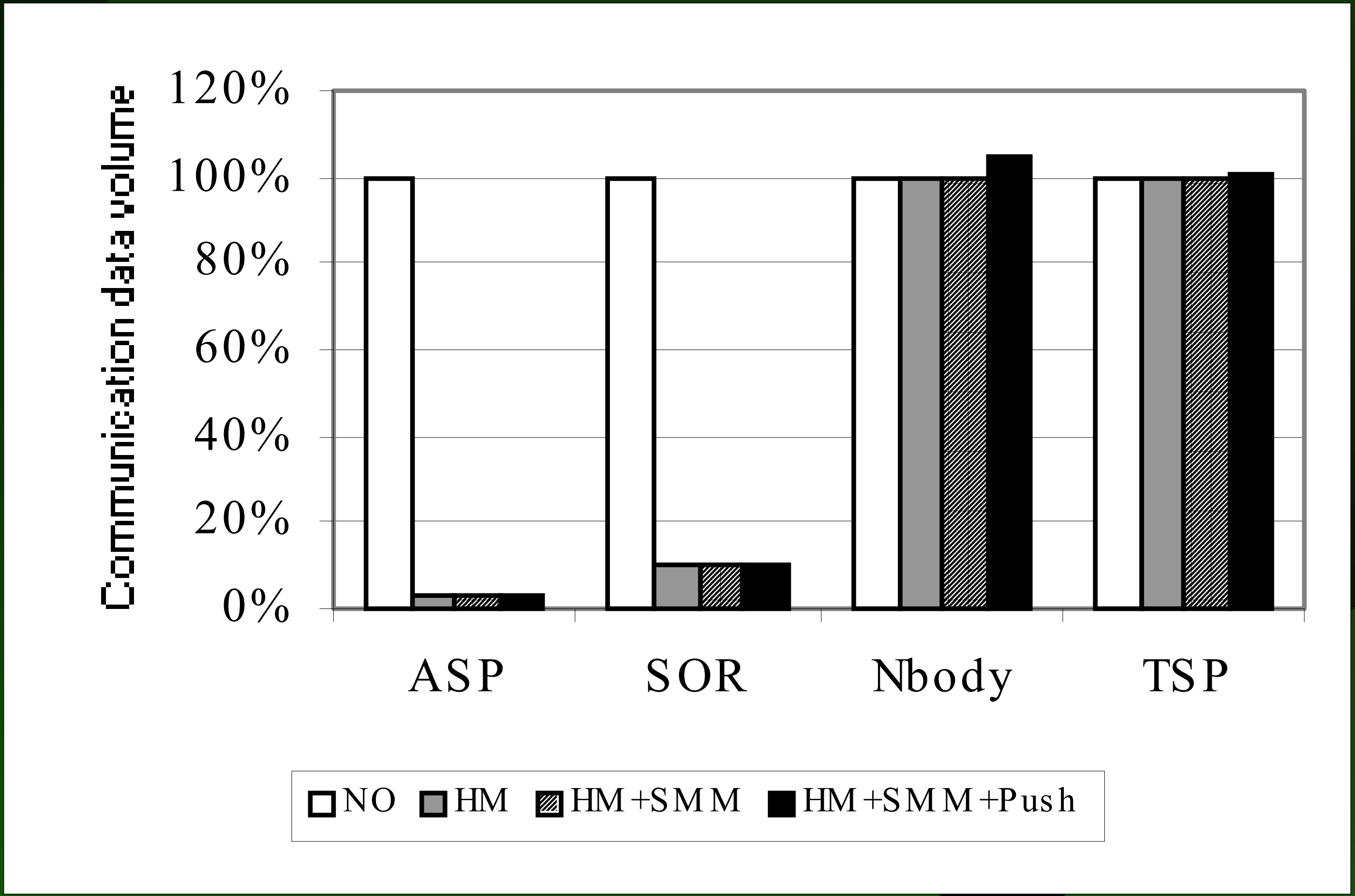
Effect of Optimizations – Breakdown of execution time



Effect of Optimizations – Message number



Effect of Optimizations – Communication data volume



Conclusion

- Global object space for distributed JVM
- Distributed-shared object
  - More efficient cache coherence protocol and garbage collection in distributed JVM
  - Facilitate further optimizations in GOS
- Effective runtime optimizations in GOS
  - Object home migration
    - Single writer access pattern
  - Synchronized method migration
    - Non-home execution of synchronized method of DSOs
  - Object pushing
    - Small size object graph

Future work

- Incorporate DSO with distributed garbage collection
- More adaptive cache coherence protocol that automatically adjusts to various object access patterns in GOS

Q & A