High Performance Java Remote Method Invocation for Parallel Computing on Clusters

Guillermo L. Taboada*, Carlos Teijeiro, Juan Touriño

computer architecture group

UNIVERSIDADE DA CORUÑA
SPAIN

taboada@udc.es

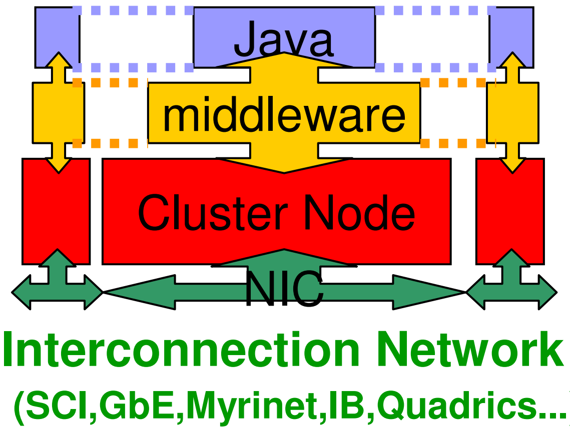
IEEE Symposium on Computers and Communications (ISCC'07), Aveiro (PT)

Outline

- Introduction
- Designing Java RMI Optimization
- Implementation: Opt RMI
  - Transport Protocol Optimization
  - Serialization Overhead Reduction
  - Object Manipulation Improvements
- Performance Evaluation
- Conclusions

Introduction (I)

- ↑ interest on clusters (↑ comput. ↓ cost)
- Growing solution:
  - Java (and HPC Java) on clusters
- Challenge: scalable peformance cluster+Java
  - Network performance is scalable
  - Java middleware less efficient than native code, especially Java RMI
    - → Java is not going to scale performance
    - High Performance Networks not supported or supported with poor performance
      - Ways of support:
        - IP Emulations
        - High Performance Sockets



Introduction (II)

- Target platform: High-speed Network Clusters
  - High-speed networks + associated software libraries play a key role in High Performance Clustering Technology
  - Diferent technologies:
    - Gigabit & 10Gigabit Ethernet
    - Scalable Coherent Interface (SCI)
    - Myrinet, Myrinet 2k, Myri-10G (10GbMyrinet & 10GbE)
    - Infiniband
    - Qsnet, Giganet, Quadrics, GSN - HIPPI
  - Small hw latencies (1.3-30us)
  - High bandwidths (>= 1Gbps)
  - Experimental results presented on Gigabit Ethernet and SCI

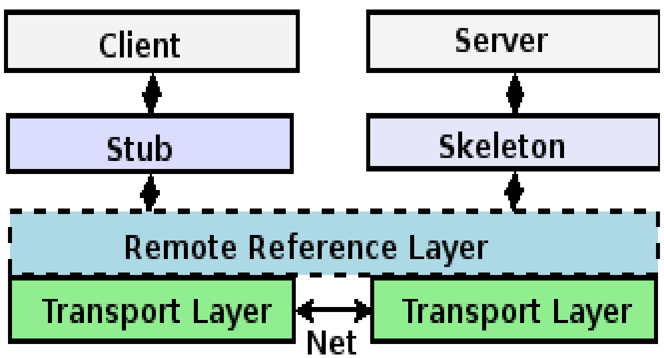
Introduction (III)

- Java RMI on Clusters
  - Java RMI is a framework for developing parallel and distributed applications. It's a higher level solution compared to sockets programming, allowing for rapid development.
  - But… inefficient protocol on clusters, showing high latencies
    - Considerable inefficiency on high-speed network clusters
    - Java's portability means in networking only TCP/IP support
    - High-speed network clusters use (inefficient) IP emulation libs.
      - SCIP, ScaIP, IPoGM, IPoMX, IPoIB

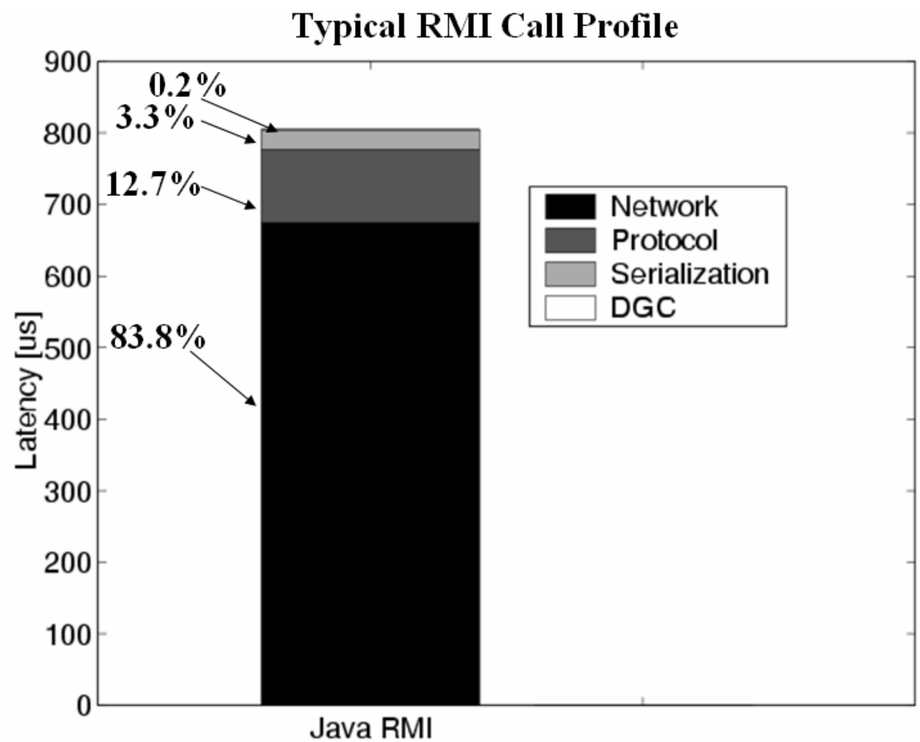
Introduction (&IV)

- Java RMI on clusters. Optimization projects:
  - **KaRMI** (JavaParty/Univ. Karlsruhe). RMI replacement for clusters. Good performance with small transfers and Myrinet support
  - **RMIX**. (Emory Univ. Atlanta) RMI extension including new communication protocols, but still inefficient on High-speed clusters (oriented to semantic protocols)
  - **Manta**. (Vrije Univ. Amsterdam) Java to native code compilation. Myrinet support.
  - **Ibis**. (Vrije Univ. Amsterdam) RMI extension for grid computing. Myrinet support.

Designing Java RMI Optimization (I)



- **Java RMI layered architecture**



- **Profiling 3KB Object call on SCI**

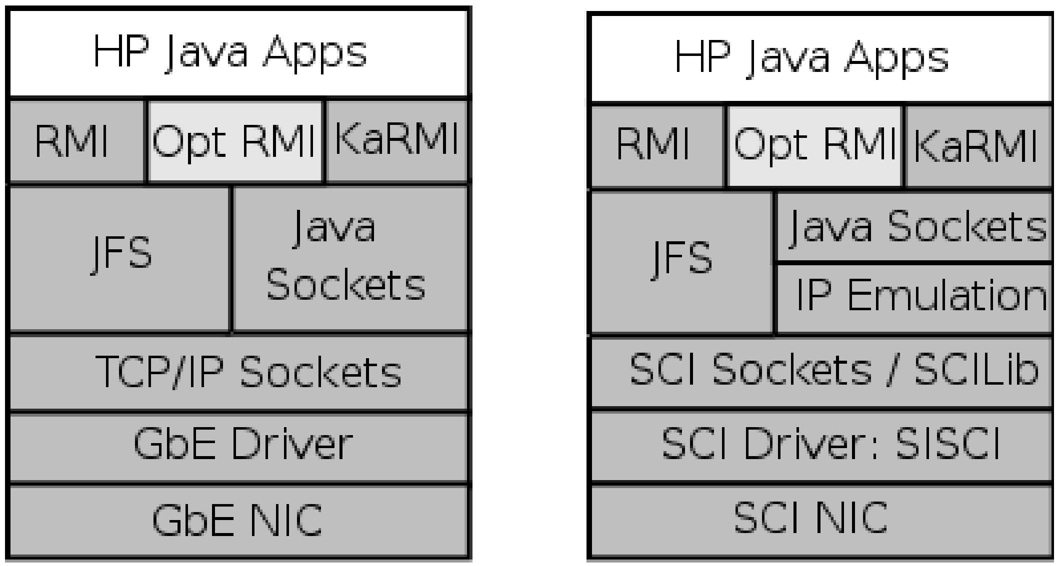
Designing Java RMI Optimization (II)

- **Java RMI Optimization tailored for High performance Java parallel applications on Clusters:**
  - **Restricted to the most typical configuration in a cluster**
  - **Goal: higher performance with little tradeoffs**
  - **Assumptions:**
    - **Shared file system for class loading**
    - **Homogeneous architecture of compute nodes**
    - **Use of a single JVM version**

Designing Java RMI Optimization (&III)

- **Java RMI Optimizations**
  - **Transport Protocol Optimizations**
    - ***High Performance Sockets Support***
    - ***Block-data information reduction*****: minimizing block-data control in serialization. Avoid block-data buffering for serialized data**
  - **Serialization Overhead Reduction**
    - ***Native Array Serialization.*** **A high-performance sockets implementation allows for sending primitive data types directly**
  - **Object Manipulation Improvements**
    - ***Versioning Information Reduction*** **(description of serialized class)**
    - ***Class Annotation Reduction*** **(class location)**
    - ***Array Processing Improvement***

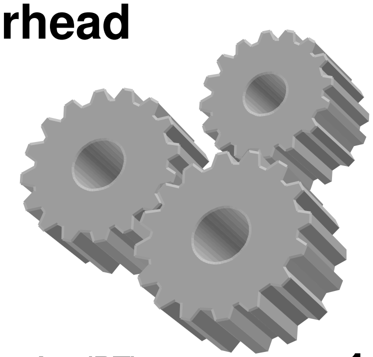
Java RMI Parallel Application Stack



- **Software architecture overview**

Transport Protocol Optimization (I)

- **High Peformance Sockets Support with Java Fast Sockets (JFS):**
  - **1st High Performance Java Sockets implementation**
  - **High Performance Network libraries support**
    - **Through native libraries on SCI, MX & native Sockets**
  - **Implements an API widely spread (Java Sockets)**
  - **Avoids the use of IP emulations (less efficient protocol for error-prone environments, with several layers)**
    - **Numerous libraries → ↑ communication overhead**



Transport Protocol Optimization (&II)

- **Java Fast Sockets (JFS) implements Java Sockets API in a way:**
  - **Efficient & portable through:**
    - **general "pure" Java solution**
    - **Specific solutions that access native communication libraries (SCI Sockets), reducing data copies**
    - **The fail-over approach applied to the selection of libraries: the system tryes to use native communication libraries with higher performance. If this is not possible, JFS uses the "pure" Java general solution**
  - **User transparency:**
    - **Setting *JFSFactory* as the default Sockets Factory in a small launcher application with Socket.setSocketImplFactory().**
      - **This application will invoke using reflection the main method. All Sockets communications wil use JFS from then on.**
        - **user@host $ java Application parameter0 …**
        - **user@host $ java jfs.runtime.RunApp Application parameter0**

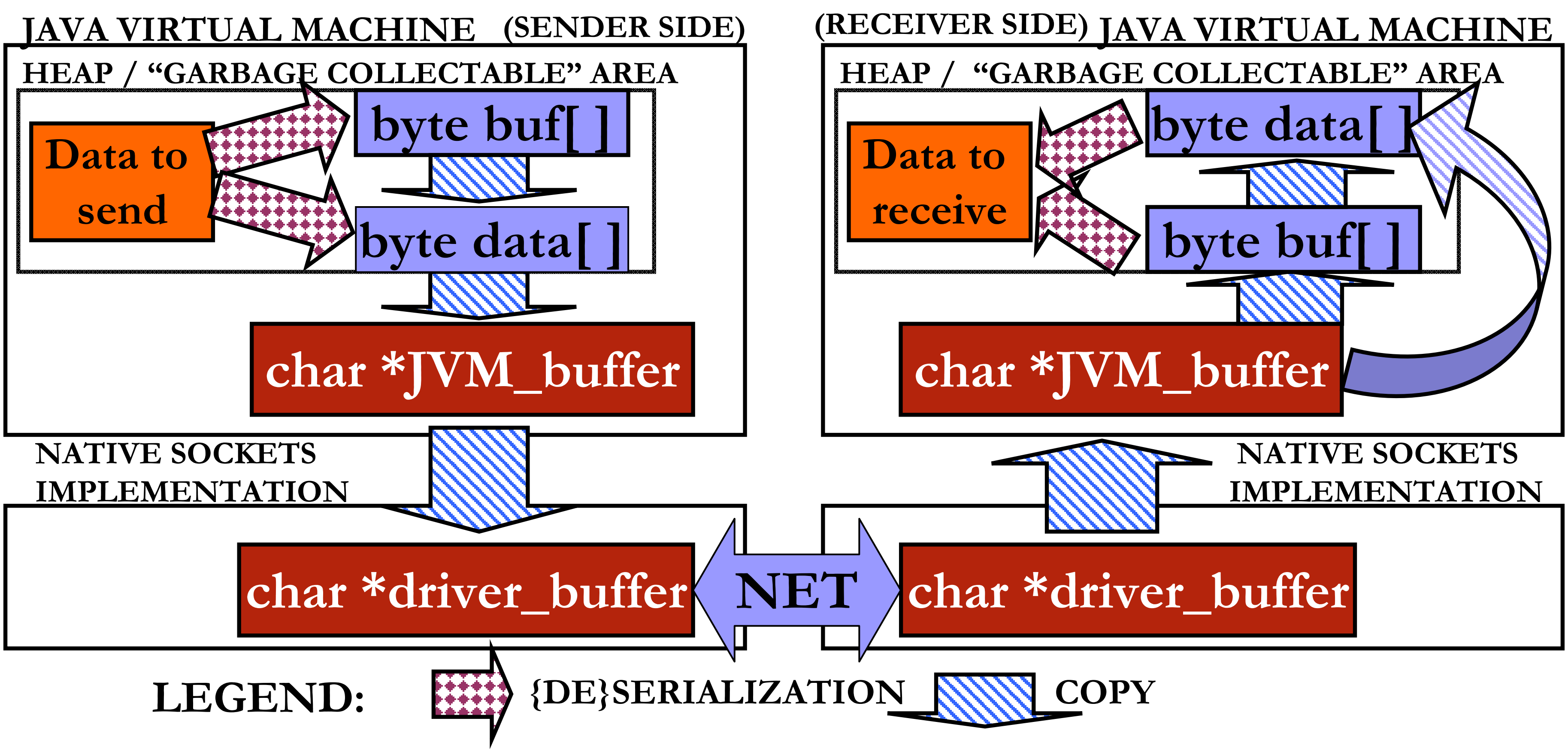
Serialization Overhead Reduction (I)

- **Java Sockets restriction: sending only byte[]**
  - **Primitive datatype arrays have to be serialized**
    - **Optimized in Java for int[] and double[] (native serialization)**
    - **JFS avoids serialization by throwing away the restriction!**

JFSOUTPUTSTREAM
+ write(int array[])
+ write(long array[])
+ write(double array[])
+ write(float array[])
+ write(short array[])
+ write(ByteBuffer directBB, int position, int size)
+ write(Object array, int pos, ByteBuffer directBB, int init, int size)

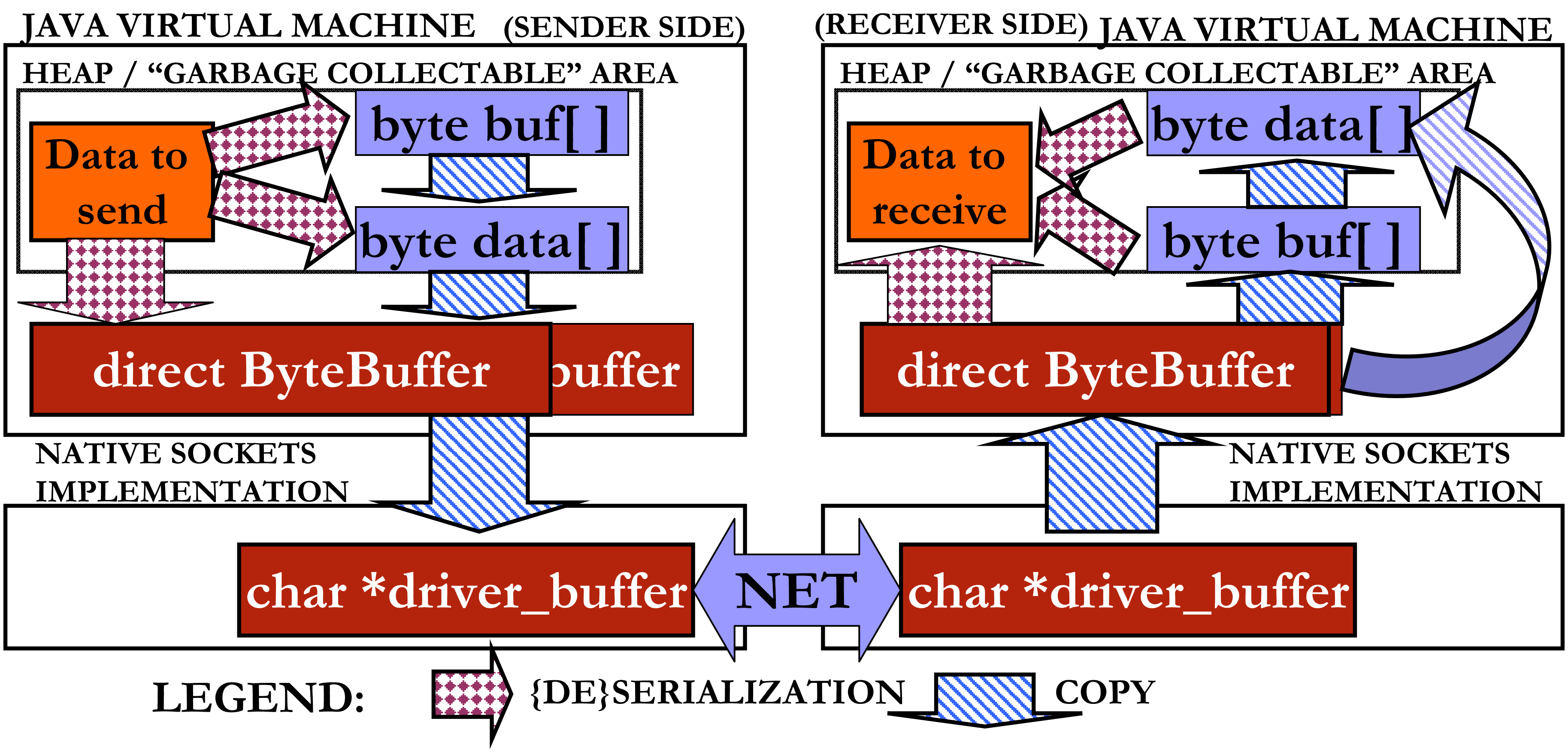
Serialization Overhead Reduction (II)

- **Default scenario in Sun's Java Sockets communication**



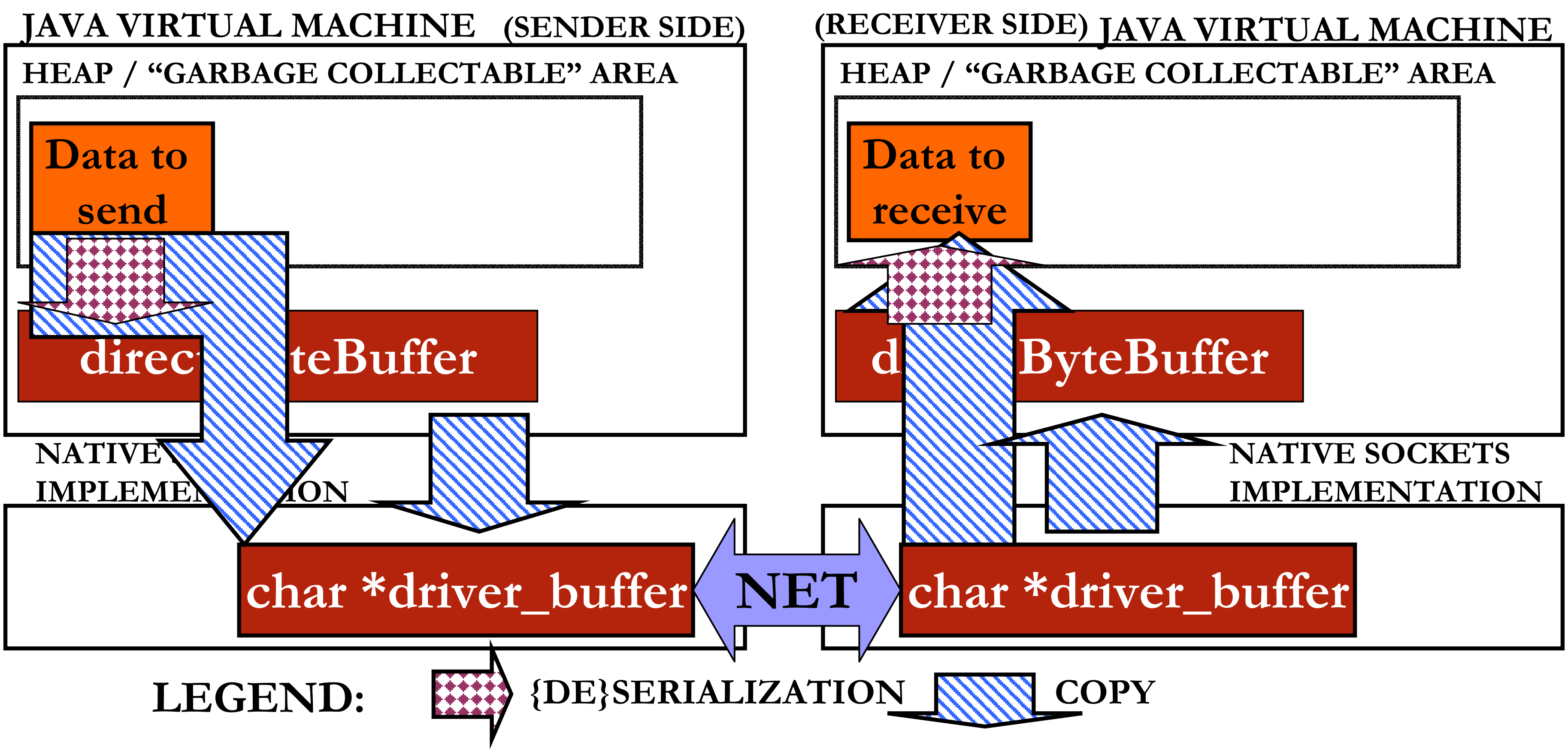
Serialization Overhead Reduction (III)

- **JFS communication using Java NIO direct ByteBuffer**



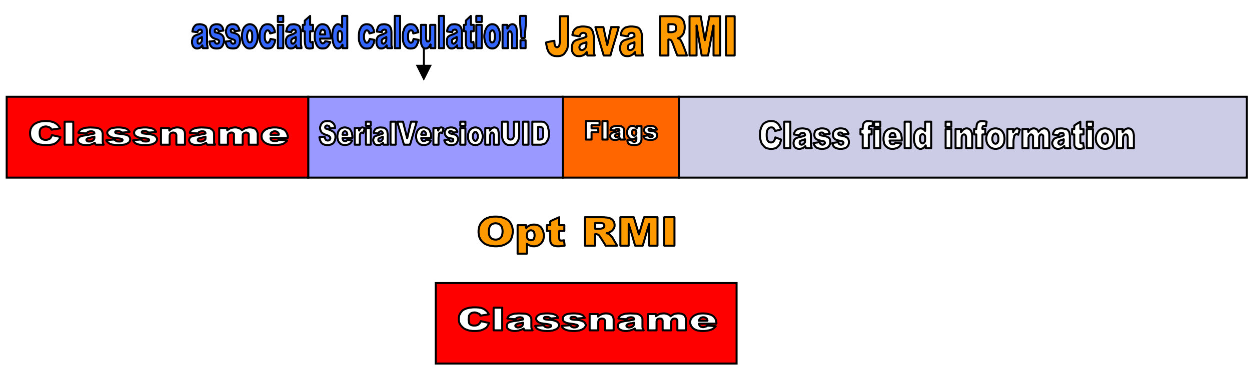
Serialization Overhead Reduction (&IV)

- **JFS zero-copy communication. Avoids copying and serialization**



Object Manipulation Improvements (I)

- **Versioning Information Reduction**
  - **Send only the class name. Important payload reduction.**
  - **With a shared file system + single JVM reconstruction is possible**



Object Manipulation Improvements (&II)

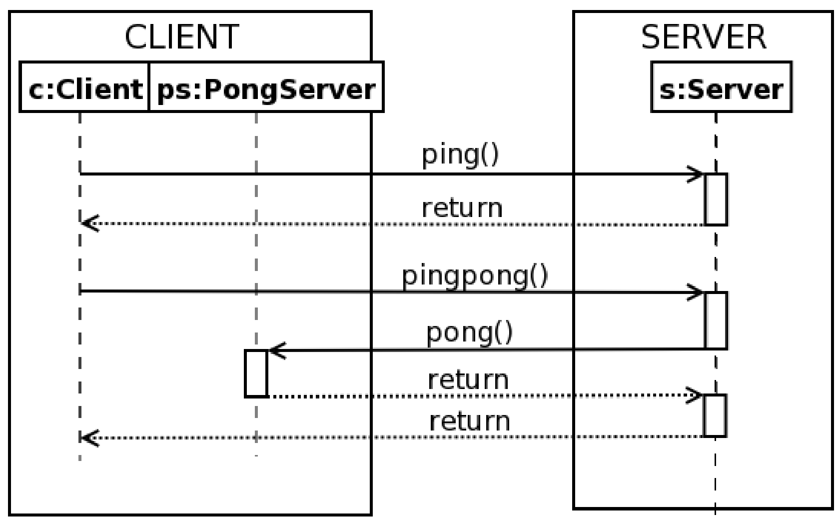
- Class annotation reduction
  - Location (String) to load a class object from
  - With a single JVM it is guaranteed that java.* classes can be loaded by the default class loader
  - Avoid serialization of java.* class names
- Array processing improvement
  - Common communication pattern in parallel applications
  - By default arrays are handled as generic objects
  - Specific method for dealing with arrays
    - Early detection of arrays (cast)
    - Optimized data type checking (common case first)
    - JFS array type processing (avoids serialization and “extra copies)

Performance Evaluation (I)

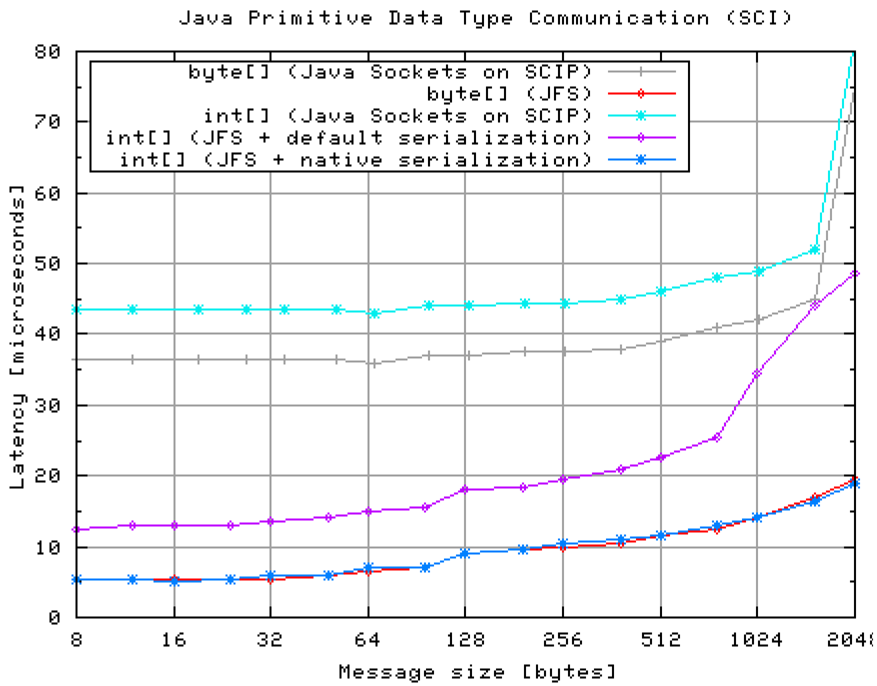
- **Experimental configuration:**
  - PIV Xeon at 3.2 GHz 2GB mem (hyperthreading disabled)
  - SCI (Dolphin), GbE (Intel PRO/1000 MT 82546 GB)
  - Java: Sun JVM 1.5.0_05
  - gcc 3.4.4
  - Libraries:
    - SCI SOCKET 3.0.3
    - DIS 3.0.3 (IRM/SISCI/SCILib/Mbox)
    - KaRMI 1.07i
  - Linux CentOS 4.2 kernel 2.6.9

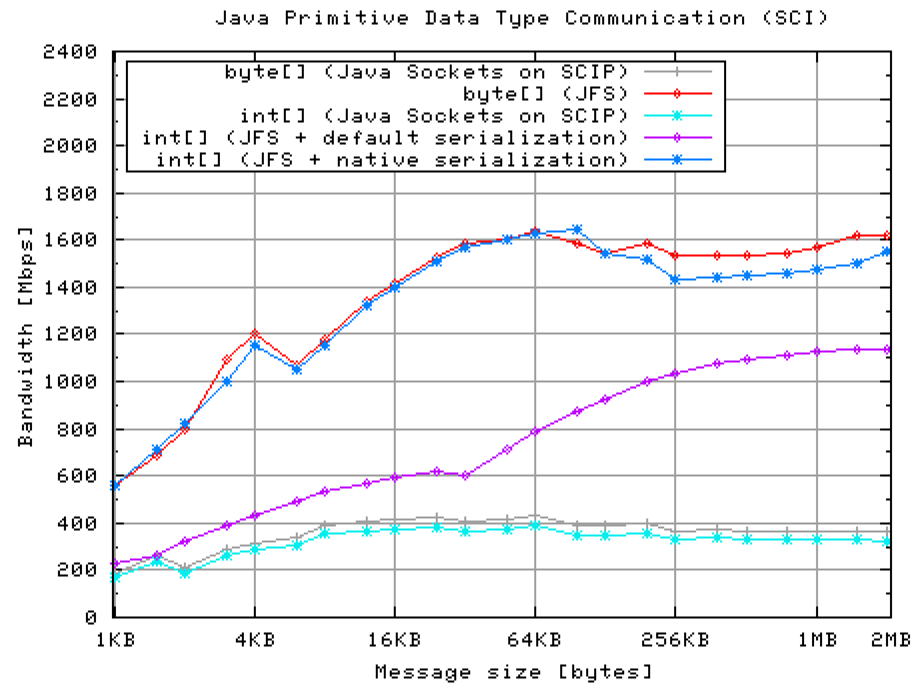
Performance Evaluation (II)

- **Benchmarking**:
  - NetPIPE Java RMI and Java sockets
    - Ping and ping-pong test
    - Java Just in Time (JIT) compiler (warm-up 10000iter.)



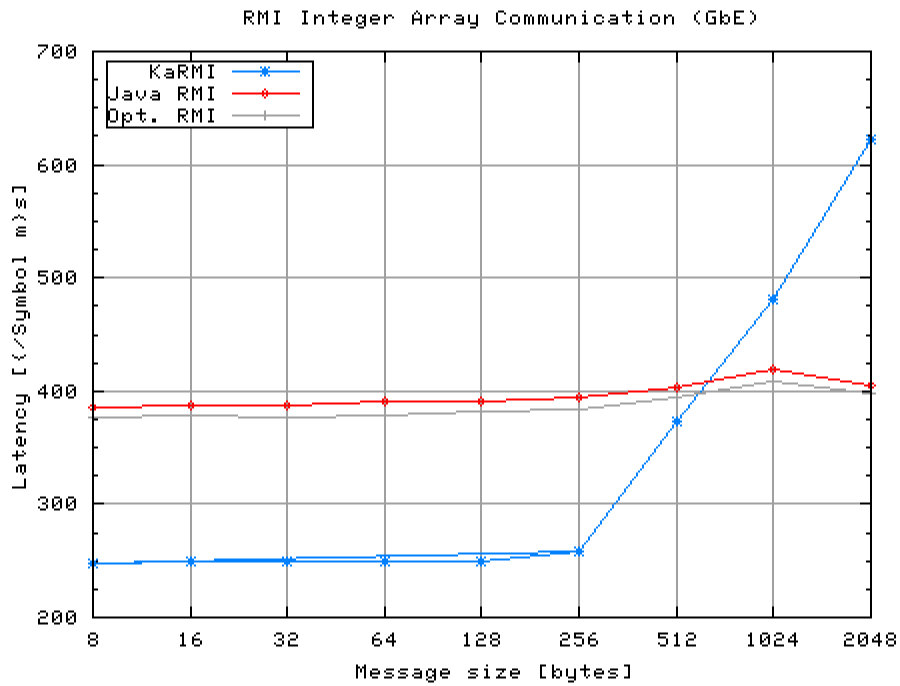
Performance Evaluation (III)

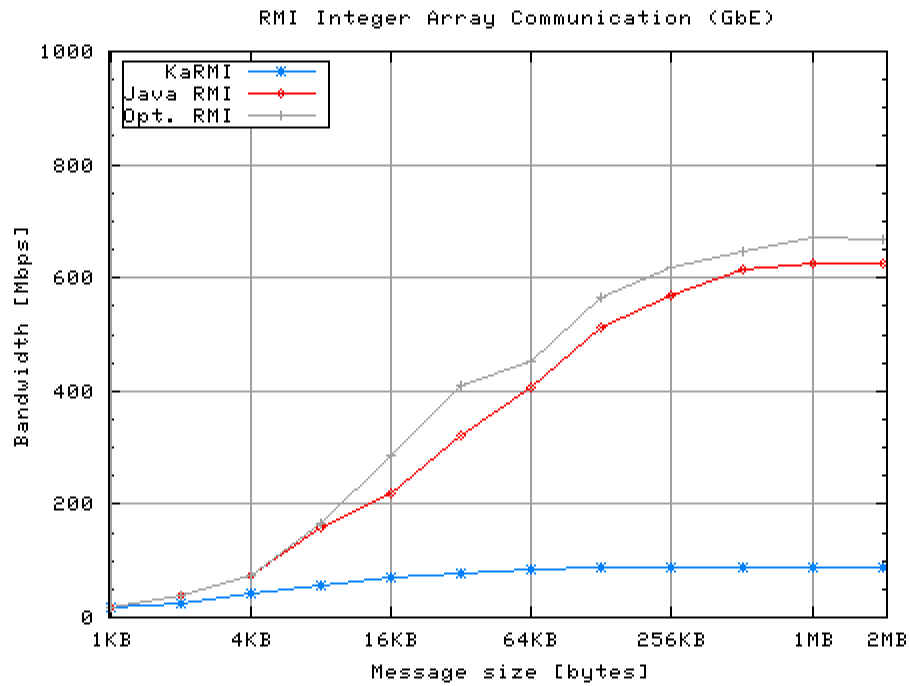




- **JFS can avoid native serialization -> sending int[] is the same as byte[]**
- **JFS avoids TCP/IP processing (Java Sockets not, SCIP) and “extra” copies**

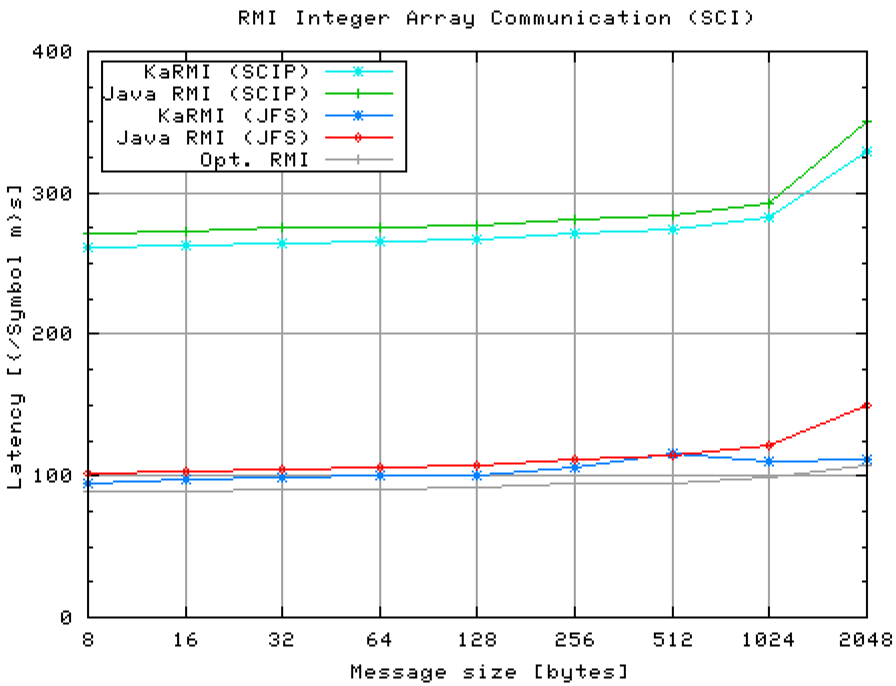
Performance Evaluation (IV)

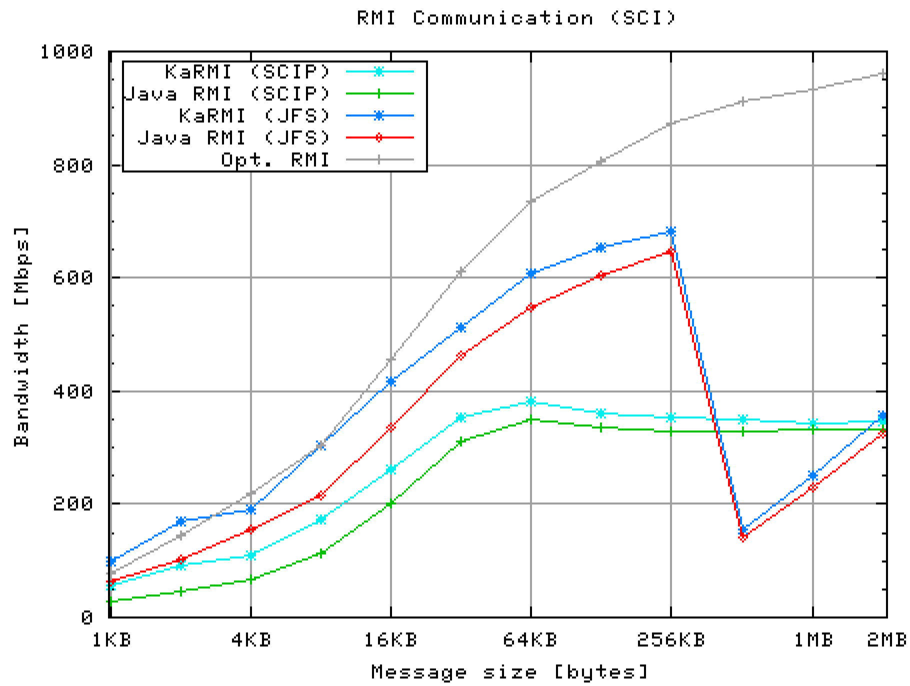




- **KaRMI shows low latencies but also low bandwidths.**
- **Opt. RMI and Java RMI results are similar for short messages, and for long messages Opt. RMI slightly outperforms Java RMI**

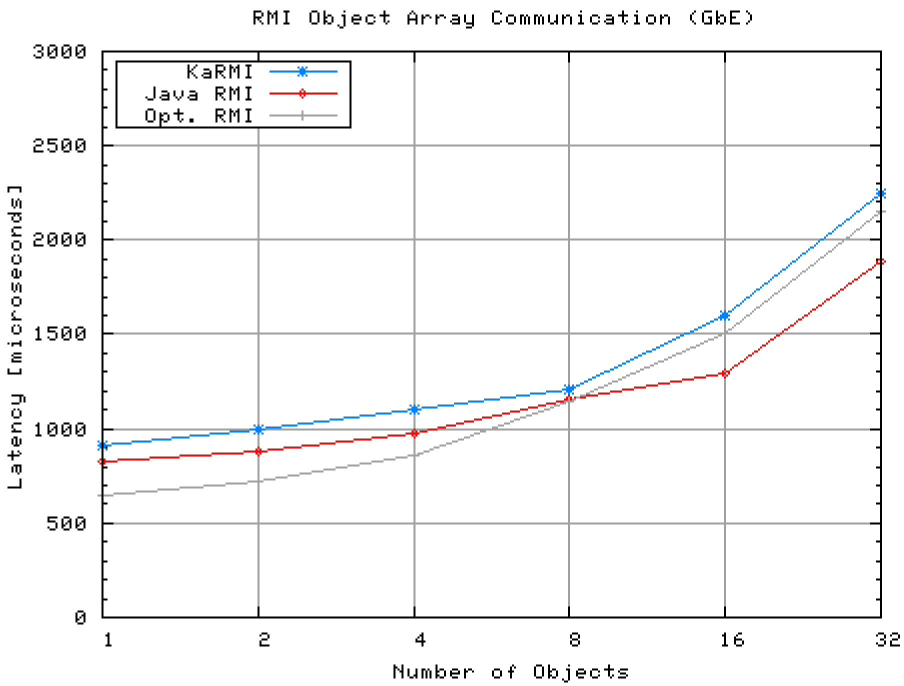
Performance Evaluation (V)

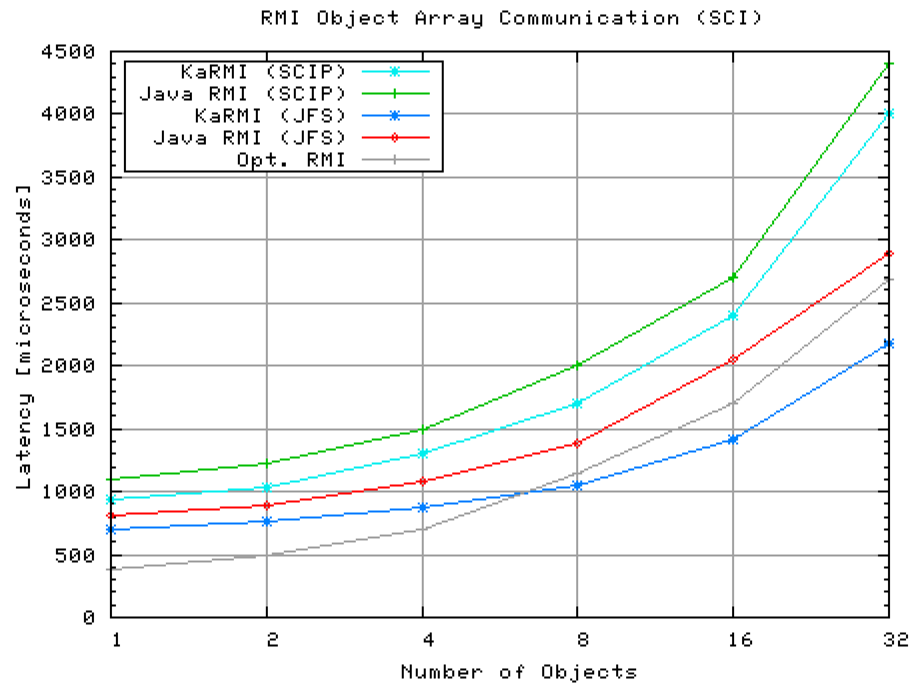




- **KaRMI performs much better on SCI. It has been designed with high performance libraries in mind.**
- **SCIP is not competitive as transport layer**
- **Opt. RMI improve performance for long messages**

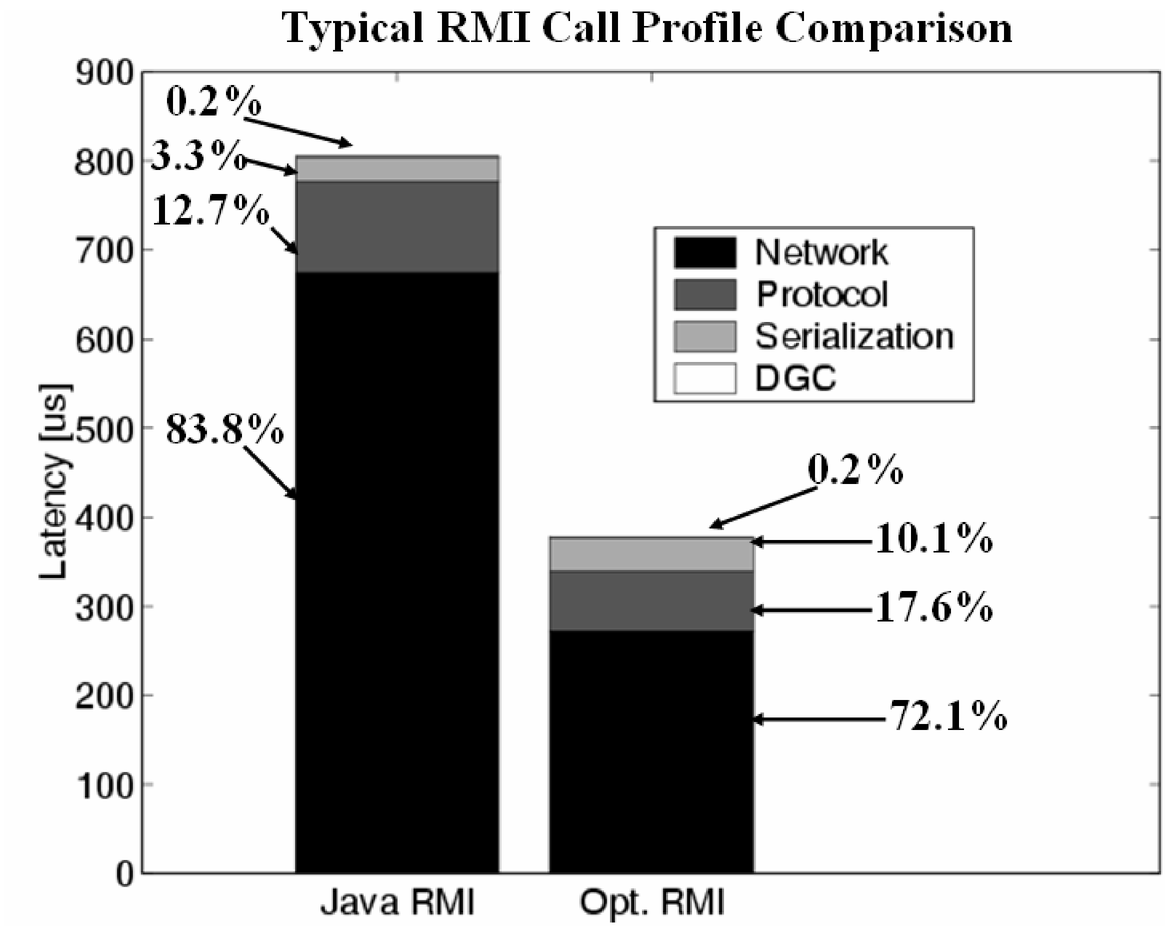
Performance Evaluation (VI)





- **Opt RMI. optimizes RMI calls with small number of objects. Sending 1 object the most common case!**

Performance Evaluation (VII)



- **Profiling of a 3KB Object call on SCI**

Conclusions (I)

- Presented a more efficient Java RMI implementation (Opt RMI)
  - Transparent to the user
  - Interoperable with other systems
  - No source code modification
  - Widely spread API
- Opt RMI protocol tailored for high-speed clusters
  - Basic assumptions about the target architecture reduce protocol overhead (trade-off interoperability vs. performance)
  - Optimizing the "most common case" for parallel computing: primitive datatype arrays
  - Implementing the protocol on top of Java Fast Sockets (JFS)
    - Avoiding serialization
    - Reducing unnecessary copies
- Protocol optimizations focused on:
  - Reducing block-data information
  - Reducing versioning information
  - Reducing class annotations

Conclusions (&II)

- The Opt RMI protocol reduces RMU call overhead, mainly on high-speed interconnection networks and for common communication patterns in Java parallel applications
- Experimental results on Gigabit Ethernet and SCI have shown significant performance increase, both for basic data type arrays and objects

High Performance Java Remote Method Invocation for Parallel Computing on Clusters

Guillermo L. Taboada*, Carlos Teijeiro, Juan Touriño

UNIVERSIDADE DA CORUÑA
SPAIN

taboada@udc.es

IEEE Symposium on Computers and Communications (ISCC'07), Aveiro (PT)