Enabling unmodified applications to run on a cluster

**dJVM: Implementing a Distributed Java Virtual Machine**

**Arrin Daley**

A Java Virtual Machine is a piece of software used to execute Java applications. A virtual machine is able to present a convenient view of the underlying hardware. This allows virtual machines to create the illusion of a single computer from a cluster (a network of physically separate, distributed computers). This gives access to the computing power of a cluster, with the ease of use of a single computer.

**dJVM** modifies the Jike's Virtual Machine to virtualise a cluster. Previously distributed virtual machines have been created and are then used to research different optimisations to achieve better performance. This approach means that fundamental design decisions have already been made. dJVM is a virtual machine which aims to explore the tradeoffs of these design decisions by implementing a system capable of multiple configurations allowing designs to be compared and evaluated in a single Distributed Virtual Machine (DVM). Some of these design choices and how dJVM implements these are detailed below.

### Updating, Invalidation and Hybrid Consistency

The one of the most performance critical of the design choices is the mechanism for providing consistency of shared memory. There are two basic approaches:

- **Invalidation**
  - Broadcast: Bitmap of writes
  - Node 0
  - Node 1
  - Node 2
  - Node n

- **Updating**
  - Broadcast: Bitmap of writes and the new values.
  - Node 0
  - Node 1
  - Node 2
  - Node n

These approaches are optimal for different memory access and sharing patterns, which vary according to the application being run. dJVM is able to use both invalidation and updating so the consistency scheme can be changed to suit the target application, hardware, or network loads.

dJVM allows the development of hybrid consistency models that are able to update selected writes whilst using invalidation on others. This is particularly relevant when dealing with locks, which designate areas where writes are performed on shared objects. These regions are therefore a very good candidate for updating.

### Homeless Consistency

Invalidation systems typically have a home node for each piece of data, this home node has the advantage of always being able to access the data locally without fetching. This is a significant advantage and makes it worthwhile migrating the home to the node in the cluster where it will be used most (object migration). Object migration involves looking at recent accesses and based on these making a decision about whether the home should be migrated. Migration will often be either too reactive or not reactive enough, is difficult to implement and can require tuning for individual applications. dJVM uses homeless consistency which allows the data to be migrated automatically to the node which performed the latest write. This optimises most access patterns and is able to minimize consistency messages. Homeless consistency is also more flexible and can be used to implement home-based consistency if required.

**Finally...**

dJVM is able to run Java applications without any modifications on a cluster. Unlike previous Distributed Virtual Machines, dJVM is able implement all of the major approaches to maintaining consistency using a homeless consistency model and homeless locking. dJVM has been able to implement invalidation and updating to provide an apples to apples comparison and also allows the two to be merged to create several hybrid consistency schemes. As such dJVM forms a solid base for further research into the tradeoffs involved in the design of distributed memory and allows the exploration of previously unconsidered hybrid consistency schemes.

Joint work with Ramesh Sankaranarayana & John Zigman