A. Arpaci-Dusseau **Department of Computer Science**
CS739: Distributed Systems **University of Wisconsin, Madison**
section*

Towards Transparent and Efficient Software Distributed Shared Memory – SOSP'97

1 Introduction

- What is the appeal of programming with shared memory (compared to message passing)? What are the drawbacks?
- What is the motivation for Shasta? (i.e., what is wrong with previous DSM approaches?)
- At a high-level, what are the challenges of transparently executing existing binaries? At a (very) high-level, what is the approach of Shasta?
- What does Shasta assume about their environment? What workloads do they target?

2 Basic Design of Shasta

- What are the basic data structures that Shasta maintains to support distributed shared memory?
- Shasta adds protocol code for most read operations. What is the simplest case for a read? What are the different states that the data can be in and how does that impact the protocol?
- What happens on a write? (Can the optimization with checking the flag value instead of the state table be used for writes?)
- Shasta explicitly polls the network so that a process can handle a message. What are the pros and cons of polling (versus using interrupts)?
- Shasta wants to be able to run on clusters of SMP nodes. What is wrong with the previous protocol when run on an SMP node? One way to fix this problem is to add synchronization around the Shasta protocol; why isn't this a good idea? How do they fix the problem instead?

3 Fully Supporting an Instruction Set Architecture

- To provide full transparent execution of binaries, the developers of Shasta felt it was essential to support instructions like load-locked and store-conditional (which are used to implement synchronization). How are the load-locked and store-conditional instructions used to implement a binary lock (Figure 1)? How are these instructions often implemented in hardware and why doesn't Shasta emulate this implementation? Shasta makes the observation that the state of the line (invalid, exclusive, shared) can be used to indicate whether the store-conditional should succeed; how is this done?

- Background: Different architectures support different consistency models. What is *strict* consistency? What is *sequential* consistency? What is *processor* consistency? What is *weak* consistency? What is *release* conistency?

- Why are page-based DSM systems unable to provide the consistency model required by most architectures? What does the memory barrier (MB) instruction in the Alpha ensure? What are the allowable values for r1 and r2 in the code of Figure 2? How does Shasta ensure the Alpha memory model? How does it implement sequential consistency?

4 Providing OS Functionality

- Why is invoking a system call a special case? How does Shasta validate data for system call arguments?

- Shasta wants to support applications that have a dynamic number of processes over its lifetime. What is the concern here? What is their solution?

5 Performance Results

- What size cluster do they run on? What is their network?

- What do they show in Table 1? Table 2?

- How significant are the SPLASH-2 applications? What type of performance do they see with Shasta?

6 Conclusions

- Do you think DSM is a good match for the distributed systems of today?