Technical Report #1367, EE Dept., Technion, Isreal, June 2003.

In-Kernel Integration of Operating System and Infiniband Primitives for High Performance Computing Clusters: a DSM Example

Liran Liss, Yitzhak Birk and Assaf Schuster
Technion – Israel Institute of Technology
{liranl@tx, birk@ee, assaf@cs}.technion.ac.il

Abstract

The Infiniband (IB) System Area Network (SAN) enables applications to access hardware directly from user level, reducing the overhead of user-kernel crossings during data transfer. However, distributed applications that exhibit close coupling between network and OS services may benefit from accessing IB from the kernel through IB's native Verbs interface, which permits tight integration of these services. We assess this approach using a sequential-consistency Distributed Shared Memory (DSM) system as an example. We first develop primitives that abstract the low-level communication and kernel details, and efficiently serve the application's communication, memory and scheduling needs. Next, we combine the primitives to form a kernel DSM protocol. The approach is evaluated using our full-fledged Linux kernel DSM implementation over Infiniband. We show that overheads are reduced substantially, and overall application performance is improved both in terms of absolute execution time and scalability.

1. Introduction

Infiniband (IB) [1] is a high-performance SAN architecture that implements in hardware legacy software protocol tasks such as reliability and multiplexing among different connections. New hardware capabilities such as Remote Direct Memory Access (RDMA) are also supported. Applications can send and receive data at high rates when accessing IB through user-level networking interfaces, e.g., VIA [2]. However, since IB defines its basic primitives in the kernel, kernel subsystems and extensions can also exploit the new hardware.

In this paper, we assess the benefits of accessing IB through the kernel for applications that exhibit close coupling between network services and those of the operating system. We use a software Distributed Shared Memory (DSM) system as a context.

DSM is a runtime system that emulates shared memory across a computing cluster [3,4]. Software DSMs implement an invalidation-based protocol using the operating system's page protection mechanism. Access rights to invalidated pages are revoked, while a page fault triggers a protocol action that updates the page.

Software DSM protocols vary widely. Some tolerate the coarse sharing granularity induced by the OS/hardware (the system page size) by using relaxed consistency memory models (e.g., Lazy Release Consistency (LRC) [4]), while others employ fine-grain sharing and retain the intuitive Sequential Consistency (SC) memory model [5,6]. However, several observations hold for DSM protocols in general:

- *Each protocol invocation requires at least one system call.* These are usually multiple calls for changing page protection or for synchronizing with application or communication threads (using semaphores, mutexes, etc.).
- *The communication is inherently asynchronous.* Various request messages (Pages, Locks, Diff applications, Barriers) arrive unexpectedly.
- *Latency is important.* A DSM system is intended for parallel, computation-bound applications. An application thread waiting for a remote response can severely affect the parallel computation. In addition, the communication workload comprises mostly small packets, so high bandwidth does not suffice.
- *Application data is frequently transferred among nodes.* This data is not processed by the DSM protocol, and its destination address is known in advance.

Therefore, reducing expensive system calls and user-kernel crossings, high responsiveness to asynchronous events, and efficient data transfer in terms of buffer copies and associated OS protocol processing are all required for high performance.

The introduction of high-performance user-level SANs to DSM systems [7,8] eliminated OS protocol processing, and reduced extra memory copying through remote memory operations. Responsiveness, however, remains a problem: constant polling is the most responsive method, but wastes valuable CPU cycles; a separate communication thread requires a context switch to and from it; catching a signal depends on the receiving task being scheduled. Also, memory protection system calls are reported to constitute substantial overhead in user-level implementations [9,10]. Accordingly, DSM systems appear well suited for evaluating the kernel/IB platform.

Previous work demonstrated the advantages of integrating the kernel network protocol stack (TCP/IP) with high-level protocols [11] or with the file cache [12] in network servers. In this paper, we show that this approach is beneficial even for SANs, wherein the network protocol stack is implemented in hardware.