Application Programming on a Shared Memory Multicomputer

Todd Poynor, Tom Wylegala
HP Laboratories, Palo Alto

The HP Labs MultiComputer Systems (MCS) project is investigating issues involved in writing applications for a shared memory multicomputer, defined as a set of independent computing nodes coupled through access to Global Shared Memory (GSM). MCS is an architecture specification and prototype implementation of a shared memory multicomputer. The prototype platform is based on a commercially available ccNUMA machine comprised of 4 SMP nodes. MCS leverages commodity operating systems with few or no changes; the prototype runs an unmodified Windows NT 4.0 operating system on all nodes, extended through dynamically loaded kernel modules to support multicomputer interfaces.

Shared memory multicomputers hold considerable promise as modular architectures that transcend SMP scaling bottlenecks while preserving SMP-like memory load/store programming models. Many multicomputer platforms today do not fully deliver these benefits because most resources are partitioned and shared memory is limited to inter-node message passing, as in a shared-nothing cluster. Those multicomputer platforms that we are aware of that do allow access to global resources have limited support for containing faults from propagating across nodes, leaving multicomputers at a disadvantage when compared to conventional clusters in this regard. A shared-something cluster exposes the risk that faults will propagate over the shared resource to introduce faults in other components. This issue remains a thorny problem in the industry, especially when commodity hardware and operating systems are leveraged.

The MCS project is, in part, an experiment in pursuing fault containment as strong as that of shared-nothing clusters within a shared-almost-everything multicomputer for flexibility of resource allocation and scalability. MCS global applications are a set of processes distributed across the nodes of a multicomputer that cooperate to provide a service. The applications freely employ a variety of global resources on any node. These resources are accessed in a "safe" fashion, detecting and recovering from failures and reconfiguring when the deployment of nodes and applications is changed. This comes at a price of extending commodity operating systems and customizing applications for fault containment, as well as extending commodity hardware platforms to support containment and avoid single points of failure. We examine the challenges of programming in such an environment and investigate support our platform could provide to make the programming task easier.

Consider a thread that reads the identifier of a global mutex from a GSM area, locks the mutex, updates a data structure in the GSM area, and releases the mutex. Possible disruptions in global state include: the mutex identifier may no longer be valid; the GSM hosting the mutex identifier or data structure may fail; the mutex may be found "abandoned" by the previous holder without releasing it; the set of processes and operating systems managing the GSM or mutex may change, requiring a change in global resources; and some other recovery event may be signaled by another thread, requiring the mutex be unlocked before global state may be rebuilt. Part of our task in supporting multicomputer applications is to ease the development and execution of applications in such a dynamic context.

We created a set of C++ classes to facilitate development of global services for a variety of applications and to facilitate multiple global components within one application. Among the services provided are: a framework for initiating, detecting, synchronizing, and handling global system and component failures and recovery events; component membership management, where the set of instances participating in a generation of the global component are agreed upon; and various library functions that automatically perform recovery actions.

To demonstrate our MCS prototype, we modified an existing commercial application to use global resources and to recover from certain software failures. Our primary goal was to demonstrate recovery from an OS crash on one node while shared resources were in use by the crashing node, a relatively high probability failure that exercises both kernel and application layer recovery.

MCS also targets future recovery from a number of hardware failures, including failed memory access resulting from abruptly powered off nodes or malfunctioning memory, some support for which was also prototyped in our application work. Recovery from memory failures on present-day commodity processors poses some thorny problems in regard to notifying the proper application contexts of failed memory access. On IA-64, for example, these problems include indeterminacy of notification, speculative data prefetches, and advanced instruction retirement prior to completion of memory operations. We propose a model that avoids many of these problems in that the system is not required to match a failed access with the context that issued the request. This places a greater burden upon applications to detect failures and increases the chances that an application will perform unnecessary repair work, but failures should be rare.

Our demonstration application is a Web server file cache that reduces disk I/O by caching local static files in memory (also known as a "reverse proxy"). Both the Apache and Microsoft IIS Web servers are adapted to a common MCS global Web caching component. A mixture of the two Web servers may run in the same multicomputer complex; all will share the same cache. This is not an ideal application for demonstration of multicomputer platforms, as the primary data being shared is read-only and easily replicated and partitioned across shared-nothing servers with little or no inter-node communication. The Web cache application does, nonetheless, allow us to investigate various aspects of the recovery model and performance scalability, using an easily modified technology. Failure and reconfiguration recovery scenarios include:

- A process fails while updating data structures such as hash chain pointers and cached file data reference counts. The hash chain must be rebuilt and references from failed processes cleared.
- Processes enter and leave the global application concurrent with ongoing cache access or recovery, requiring coordinated changes in the global memory areas and perhaps mutexes in use. If processes have left the global application then the associated cached files are removed from the hash table.
- GSM access may fail while building shared state, while traversing hash chains and updating cached file data reference counts and access times, and while sending cached data back to the HTTP client.

We have demonstrated that the application recovers from multiple process failures at a time in each situation where the failure could affect other processes. Application-level recovery from failure occurs within 2 seconds, which suggests competitive performance with application recovery times for a number of the leading cluster failover solutions available today. The MCS project did not reach the point where the complete system recovers from an OS crash, but several individual software components of the system have demonstrated this under prototype conditions. The prototype platform is not suitable for demonstrating recovery from hardware failures, as the interconnect cannot recover from unresponsive nodes.

Although we did not perform extensive performance tuning of the Web cache, we did characterize performance using a workload based on a popular Web server benchmark that serves static content. We consistently obtained 2X performance when resources were doubled by moving from one to two nodes with double the amount of aggregate memory for the cache. We can expect to obtain better than 2X scaling when memory is doubled on a suitable platform because the effective cache size of a shared nothing configuration is not doubled due to duplication of frequently accessed content in both caches. In our trials we saw only an insignificant increase beyond 2X scaling, caused by an excessive GSM access penalty of 12X in the prototype hardware. Experiments substituting local memory for the cache show a 40% performance improvement and linear memory scaling, suggesting that much better performance would be obtained on a more suitable platform.

In addition to researching technology issues related to platform software and hardware, the MCS program examined business issues related to the acceptance of such a platform in the commercial application marketplace, consulting with researchers and developers at a variety of potential business partners. The first major concern voiced by the ISVs was the standardization of the global APIs: (1) that the vendor of the host operating system certify the global APIs; (2) that all shared memory multicomputers based on Windows NT share the same global APIs; (3) that all shared memory multicomputers, even those based on other operating systems, share some similarity in API structure. The second major concern was the approach taken to achieve scalability across multiple nodes, which may be at odds with their existing strategy. The last major concern relates to the added difficulty of achieving high availability under the MCS failure model, a discipline not required on SMP platforms.

To help address the difficulties of attracting ISVs to multicomputer platforms, we began work on the recoverable component framework. We also planned for a developer kit, to include various global status display and modification tools, as well as application and kernel driver debugging tools adapted for concurrent, multinode debugging.

More information on our work may be found in an HP Labs Technical Report titled "Application Programming on a Shared Memory Multicomputer", to appear at http://www.hpl.hp.com/techreports/ .