Dr.Dobb's
THE WORLD OF SOFTWARE DEVELOPMENT

Building Internet Distributed Computing Systems

Our authors present a framework for harnessing distributed, tightly coupled cluster and SMP resources for computational science research.

November 01, 2005
URL:http://www.drdobbs.com/building-internet-distributed-computing/184406315

Charles teaches Computer Science at Earlham College and can be contacted at charliep@cs.earlham.edu. Joshua is a graduate student at Indiana University and can be reached at jjhursey@cs.indiana.edu. Josh works in the Cluster Computing Group at Earlham College and can be contacted at mccoyjo@earlham.edu.Vijay is a professor of Chemistry at Stanford University and can be reached at pande@stanford.edu.

The demands of computationally based research often exceed the computing resources available to any one person or, in some cases, even one institution. Leveraging "donated" cycles from machines across the Internet is one increasingly popular mechanism for providing more horsepower to such research projects. Microsoft's Jim Gray provides a thorough analysis of the favorable economics of such an approach (http://research.microsoft.com/research/pubs/view.aspx?tr_id=655).

Distributed computing projects such as SETI@home and Folding@Home use a client-server model to perform embarrassingly parallel computations, letting you tap resources—hundreds of thousands of CPUs in individual PCs throughout the world—impossible to obtain by other means. However, certain problem domains can greatly benefit from a hybrid approach: Combining the massive resources available to distributed computing projects with the tight coupling traditionally found only in supercomputers, but now often found in commodity/Beowulf clusters and modest SMP platforms. As clusters of PCs, SMP computers, and clusters of SMP computers (sometimes called "constellations") become more commonly available, it is practical to build distributed computing projects for those system architectures.

In this article, we describe a framework for harnessing distributed, tightly coupled cluster and SMP resources for computational science research. The framework combines capability discovery, load balancing, process monitoring, and checkpoint/restart services to provide a common interface to a range of cluster and/or SMP-based distributed computing resources.

We illustrate one implementation of this framework with Folding@Clusters (http://cluster.earlham.edu/project/folding-at-clusters/), a distributed computing package that uses GROMACS as its scientific core. GROMACS, an open-source molecular dynamics package (http://www .gromacs.org/), can be used to simulate the self assembly, or folding, of amino acids chains into protein molecules. The parallel version of GROMACS is built with the Message Passing Interface (MPI), a widely adopted message-passing standard with both open- and closed-source implementations available for most platforms; see http://www.mpi-forum.org/. The framework uses Cosm, the distributed computing library from Mithral (http://www.mithral.com/projects/cosm/), to perform much of the low-level work. Cosm provides you with a platform-independent interface to a variety of tasks common to distributed-computing applications.

This framework can be extended to other existing scientific cores and a range of computational research techniques; for example, computational fluid dynamics. It is worth noting that molecular dynamics, like many other areas of computational science, has been around for a comparatively long time. A significant body of tested and verified software for this and many other computational problem domains already exists. There is value in being able to reuse this software by encapsulating it within a modern distributed computing framework.

Architecture

To support the execution of scientific cores on a variety of cluster and SMP-based distributed computing resources, the framework contains the following broad areas of functionality: capability discovery, load balancing, progress monitoring, checkpointing/restarting, and results processing.

This functionality is supported by a four-part structure: the assignment/data server, mother, nannies, and children. Figure 1 shows the relationship between these components (operating-system processes) and the underlying client system.

The assignment/data server provides the client with the IP address of the data server, which provides storage for both molecules pending distribution and result sets returned by clients. The assignment/data server also provides a standard interface for scientists to submit jobs and for client systems to query for and return completed work units. All communication with the assignment/data server uses secure HTTP; internally it uses a relational database to organize work units, clients, and results.

The mother is responsible for all communication with the assignment/data server, coordination of local resources, and serves as the front end for the client system installation. The mother uses MPI to launch the nannies on the client system. After launch, the mother and nannies determine a capability profile of the system, then the mother launches children on each node as needed. There is one mother per client system installation. Figure 2 is a pseudocode representation of the mother's core algorithm.

The nanny is responsible for monitoring children on the local node. The nanny communicates with the mother via MPI and with the children via the local filesystem and operating-system signals. Signals between the nanny and child are used to determine if the child is still active, and the filesystem is used to pass checkpoint files between them. The nanny is also responsible for placement of the work unit on the local system before the child process begins,

sending the results back to the mother when the child completes, and cleaning up after the child process. The nannies do not communicate amongst themselves but rather just with the mother and the children on the local node. There is one nanny per machine/node in the client system.

The child is a thin wrapper around the scientific core. This layer is what permits the framework to adapt to a variety of existing software. The child only communicates with the mother during initialization and finalization. The child must ensure that the scientific core uses the appropriate local files and creates checkpoint files on a regular basis. Children communicate amongst themselves via MPI. MPI is used in such a way as to provide the children with a private communicator, which allows for the integration of a scientific core without requiring changes to the core's existing communication mechanisms. There are one to *N* children per machine/ node in the client system, depending on the mapping determined by the mother and the assignment/data server. The most common configuration is one child per CPU per node.

The communications structure of the assignment/data server, mother, nannies, and children is illustrated by the colored lines in Figure 1.

Capability Discovery

A Capability Profile for the client system is determined by the mother and nannies before launching the children. This profile includes: the number of nodes, number of CPUs, speed of the CPUs, CPU benchmarks (floating-point and integer arithmetic), amount of available memory, load on the machine, and bandwidth and latency of the interconnection network. Because some of this information is static, it is only determined when the initial capability discovery occurs. The other information is highly dynamic and remeasured between each work unit. Cosm provides much of this information to the nannies, which then report it to the mother. Bandwidth and latency are determined by a stripped down version of NetPipe (http://www.scl.ameslab.gov/netpipe) that is run between the mother and each nanny process (for example, each node). The mother and assignment/data server use the final capability profile to determine an appropriate work unit for the client system.

Load Balancing

Once the capability profile has been built, the mother can appropriately map child processes onto the client system. The load-balancing mechanism lets the framework use the underlying client system in the best way possible, given its capabilities and the characteristics of the current work unit. If the mother process is aware of the details of the parallel model used by the scientific core, it may be able to further refine the mapping of the child processes to use the client system more efficiently.

Progress Monitoring

The nanny monitors the progress of child processes (for example, the scientific core) on the local node via a ping-pong protocol implemented with operating-system signals. The nanny sends a signal to the child and expects a complementary signal from the child in response. If the child does not respond to the signal in a specified amount of time, the child process is determined to be either dead or stalled. When this occurs, the nanny reports it to the mother. The mother monitors the nanny processes via a heart-beat protocol implemented with MPI. The mother is able to determine if a nanny has failed by relying on the MPI layer to return an error if the remote process is not responding. If the mother is notified of a child or nanny failure, it reaps all children and nannies and starts the work unit over from the last checkpoint.

Checkpointing and Restarting

Currently, the framework requires that the scientific core have some inherent ability to generate local checkpoint data. The framework harvests those checkpoints and keeps the most recent set available in the event of a failure/restart sequence. GROMACS provides such a mechanism; however, one improvement to the framework would be the ability to collect checkpoint data independently of the scientific core. When a failure is detected, we attempt to restart the work unit with the same resource mapping. The framework provides a configuration variable that lets users control how many times a mapping is retried before the resource map is scaled back. If the work unit continues to fail, we continue to prune the mapping until we either reach a runnable state or conclude that the work unit is not stable on the particular client system. If the work unit is not stable, we notify the assignment/data server and request a new work unit.

Results

The mother detects completion of the children via the operating system return values of the children. Depending on the user-specified verbosity, the data files, log files, *stdout*, *stderr*, and other user-specified output from the scientific core are harvested from each node by the nannies and collected on the mother's node. This facilitates extraction of the results from the current work unit and the communication of those results to the assignment/data server. Figure 2, pseudocode for the mother's core algorithm, shows how capability discovery, load balancing, progress monitoring, checkpoint/restarting, and result processing are collectively managed.

Extending the Model

In keeping with our design goals, the abstractions found in the framework are applicable to many other problem spaces and technologies. The parallelism used in the model is not dependent on any specific technology; for example, Folding@Clusters happens to use MPI for its communication library. An equivalent implementation of the framework could be built using either Linda or the Parallel Virtual Machine (PVM) communication libraries. MPI-based implementations have the advantage of adapting to a variety of underlying compute resources (cluster, SMP, cluster of SMP nodes, and so on) at runtime without any additional work.

Another type of extension is using scientific cores other than GROMACS within the framework; NAMD and Gaussian are examples of such scientific cores. Additionally, the framework should work well for any embarrassingly parallel software application and any other software application with a favorable computation to the communication ratio. That is, the amount of computation dominates the amount of communication for a typical work unit. As is the case with NAMD and Gaussian, scientific cores do not have to rely on MPI to be compatible with the framework. Scientific cores could be exchanged on-the-fly between work units by having the mother replace the child processes as requested by the assignment/data server. The most challenging part of extending the framework relates to checkpointing/restarting. Whether and how a scientific core handles these two capabilities varies greatly. In many cases, scientific cores have basic checkpoint/restart services available that the framework can use appropriately. In other cases, a certain amount of additional engineering is required within the scientific core to support these services.

Tools

The particular implementation of the framework described here, Folding@Clusters, uses three software packages to implement the functionality: Cosm, MPI, and GROMACS. These combine in different ways to support capability discovery, load balancing, and the like.

Cosm, from Mithral, is a library that provides a stable, reliable, and secure environment for distributed computing. Cosm addresses two particularly important areas in our implementation of the framework: easy portability across a variety of operating systems (Linux, UNIX, Windows, and OS X) and functions necessary to build distributed computing applications. It presents a platform-independent API for services such as security, data transfer, filesystem access, and coordinated time.

MPI-2 is an extension and enhancement of the MPI-1.1 Standard by the MPI Forum. The forum was convened for the purpose of creating a standardized message- passing interface. Composed of approximately 60 people from 40 organizations from a variety of academic and commercial backgrounds, the MPI Forum serves as the ongoing steward of the library specification. There are many implementations of MPI, ranging from bare-bones research-only models to full featured, vendor-developed implementations with debugging and development tools. The framework relies on the dynamic process-spawning capabilities of the MPI-2 Standard for nanny and child process management, and MPI-level failure and recovery services.

GROMACS is an open-source molecular dynamics package built with an emphasis on performance. It relies on the MPI for parallelism and on FFTW for discrete Fourier transformations. GROMACS is one of the fastest molecular dynamics packages available. Although the primary design focus was on working with biochemical molecules such as proteins and lipids that have complex bonded interactions, GROMACS is also capable of simulating the nonbonded interactions that dominate molecular systems that are not based in biology.

Conclusion

Folding@Clusters is a framework for encapsulating existing scientific software so that it can be executed on both local and distributed cluster and SMP platforms. This framework can be applied to a variety of problem spaces; most every area of computational science has popular open- source software packages with existing parallel implementations.

DDJ

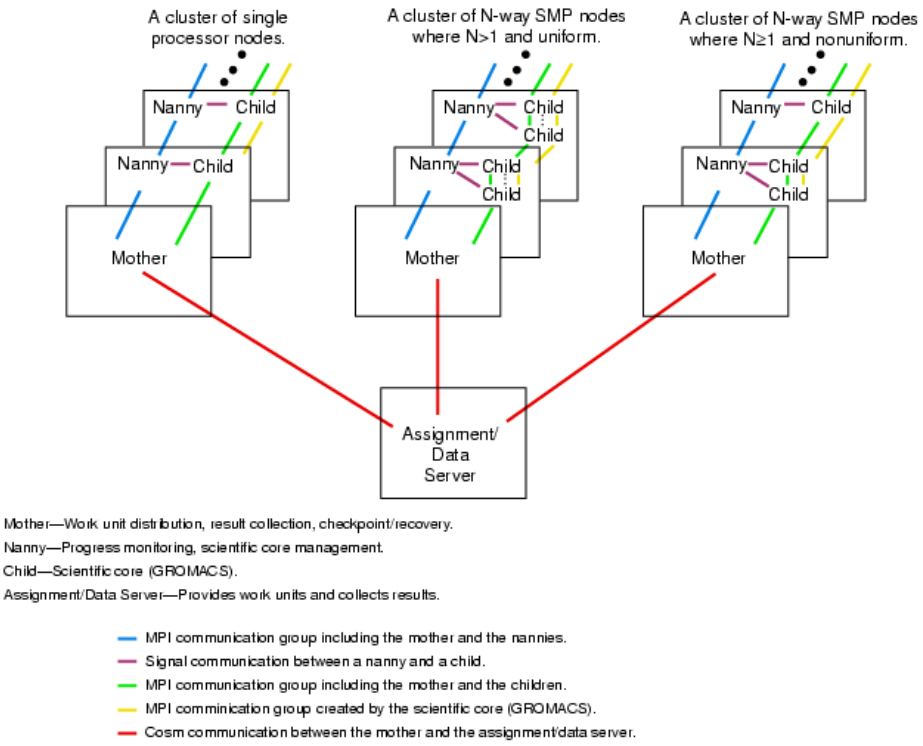

Figure 1: ***Process and communication structures.***

```
begin
process mother.conf
start nannies /* one per node in the client system */
perform capability discovery
download work unit from assignment server

while the work unit is not complete {
   prepare work unit
   start children and assign to nannies
   distribute work unit to children
   tell children to start scientific core

   while children are running {
      process checkpoint messages, transfer files
      process error messages, re-start from previous checkpoint
      if number of re-tries exceeds configuration variable stop
```

```
   }
}

send results or error to assignment server
release children
release nannies
clean up local environment
```

Figure 2: ***Pseudocode for the mother's core algorithm.***

Terms of Service | Privacy Statement | Copyright © 2016 UBM Tech, All rights reserved.