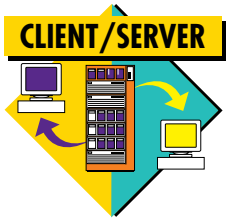
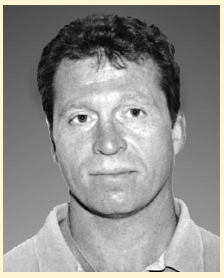


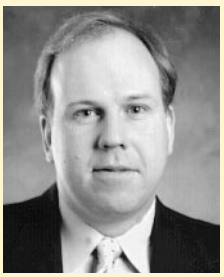
Writing Cluster-Aware Client Applications

By Steven Kohler and Thomas Casey

This article describes how application developers can use the Clinfo API routines, distributed with the HACMP for AIX software, to write "cluster-aware" client applications. Cluster-aware applications use their knowledge of a cluster's state to react intelligently to changes within that cluster. As a result, users of these client applications may not notice any change in performance or have to take any action when a change occurs within the cluster.



Steven Kohler



Thomas Casey

High availability is an essential requirement for many client/server implementations. To ensure that mission-critical data and applications are continuously available for processing, many sites have integrated high availability software into their client/server systems.

In the AIX environment, the HACMP software provides high availability services for clusters of two to eight RISC System/6000 uniprocessor and Symmetric Multiprocessor (SMP) systems and Scalable POWERparallel™ (SP) processors. HACMP allows a cluster to continue to provide data and application services critical to an installation even though a key system component—a network adapter, for example—is no longer available. When a component fails, HACMP detects the loss and shifts that component's workload to another component in the cluster. While not instantaneous, services are restored rapidly, usually within one to five minutes. The HACMP software provides recovery options for the following cluster components:

- Processors
- Networks and network adapters
- Disk and disk adapters
- Applications

When building highly available client/server systems, system engineers typically focus much planning and effort on making the server complex as fault resilient as possible. Constrained by time and money, they may overlook the client side of the "client/server" equation. Client applications, therefore, tend to be "naive." A naive client application has no knowledge of the cluster state and views the cluster as a black box that is either up or down. If a server fails, the client application typically hangs, and later must be restarted or at least reconnected to the server. Meanwhile, unable to work, stymied users ask "What's wrong? Is it my machine, the network, or what? Did I lose a lot of work?"

To build a client/server system that delivers the full benefits of high availability to its users, system engineers should address availability issues relating to client applications during the planning and implementation of that system. Clients should be cluster-aware—that is, they should be able to determine the state of the cluster and use that knowledge to react intelligently to changes within the cluster. A cluster-aware client, for example, could notify users about a problem with the cluster and instruct them to stand by while the problem is fixed. Or it could connect to an alternate server, perhaps masking the failure from the user altogether.

Writing intelligent client applications does not have to be a complicated, time-consuming task. This article demonstrates the amount of coding necessary to make client applications sufficiently robust to handle node failures transparently. We begin by briefly describing the cluster information facilities provided with HACMP. Next, we describe how the HACMP software recovers from node failure and its effect on client applications. We then describe a pair of C language routines that use functions from the Clinfo API to enable a front-end application to monitor its connection

to a Sybase® SQL Server database and, if the connection is lost as a result of a node failure, reestablish the connection to the database without having to restart the application.

Making Cluster State Information Available to Client Applications

The cluster information facilities provided with HACMP allow users to monitor the cluster state and to write intelligent client applications. These facilities include the Cluster SMUX Peer daemon, which maintains a cluster Management Information Base (MIB) that clients can access using the standard Simple Network Management Protocol (SNMP) interface, and the Cluster Information (Clinfo) daemon and libraries, which provide a simpler interface for writing cluster-aware clients.

Cluster SMUX Peer

The Cluster SMUX Peer provides SNMP support to client applications. SNMP is an industry-standard specification for monitoring and managing TCP/IP-based networks. The Cluster SMUX Peer daemon maintains a custom MIB that describes an HACMP cluster. When the Cluster SMUX Peer daemon starts running on a cluster node, it registers with the SNMP daemon, then continually gathers cluster information from the Cluster Manager daemon. The Cluster SMUX Peer daemon maintains an updated topology map of the cluster in the HACMP MIB as it tracks events and resulting states of the cluster.

Cluster Information Program

The Clinfo daemon is an SNMP-based monitor. The Clinfo daemon, running on a client machine or on a cluster node, queries the Cluster SMUX Peer daemon for updated cluster information that it stores in shared memory. Through Clinfo, information about the state of an HACMP cluster, nodes, networks, and network adapters can be made available locally to clients and applications.

Figure 1 shows the relationship between SNMP, the Cluster SMUX Peer, and Clinfo within an HACMP cluster.

Clinfo API

The Clinfo API gives the programmer a way to directly access the dynamic information about an HACMP cluster, as well as specific components within that cluster. The Clinfo API supplies both C and C++ language libraries tailored especially for this purpose. HACMP for AIX, Version 4.1 and above includes both single-threaded and multi-threaded versions of the Clinfo C and C++ API

Cluster Information Facilities

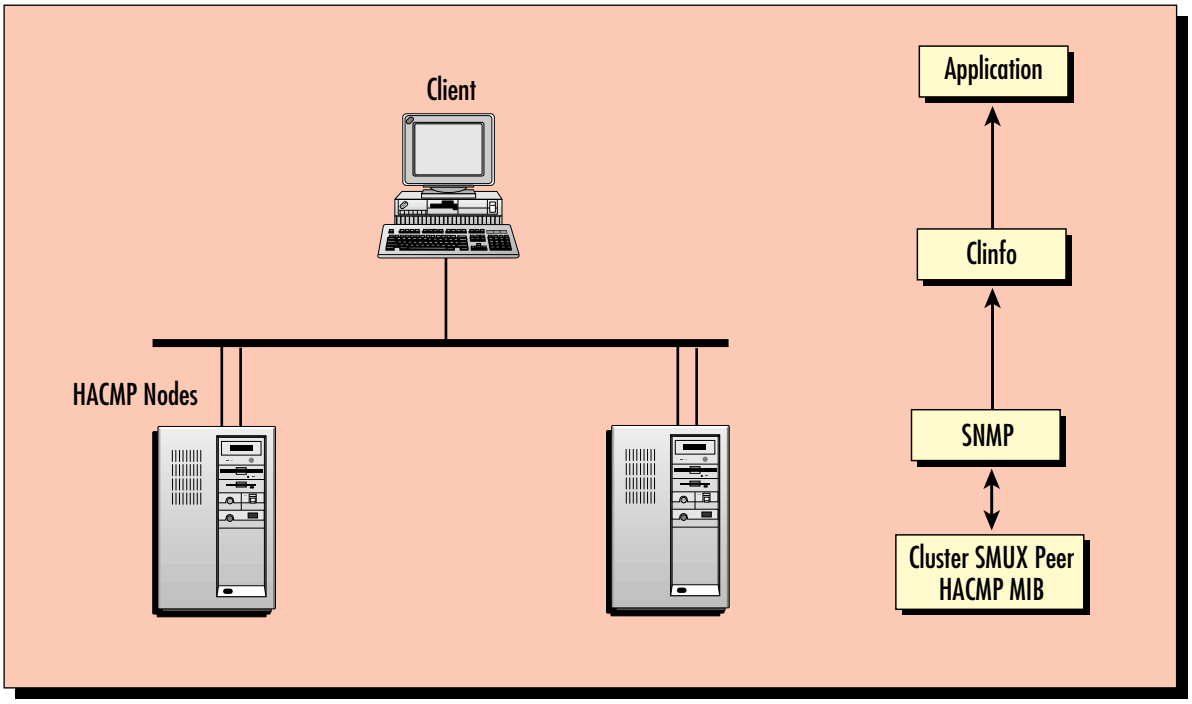


Figure 1. Cluster information facilities

libraries. The routines described in this article use the single-threaded C library.

Supported Platforms

Clinfo is supplied with the HACMP software. The source code for the Clinfo daemon and the API is distributed, so Clinfo can be ported to other machines. CLAM Associates has ported and tested Clinfo on PCs running DOS and Microsoft Windows.

Recovering from Node Failure

The HACMP software uses an agent, called a Cluster Manager, on each cluster node. The Cluster Manager monitors local hardware and software subsystems, tracks the availability of other nodes in the cluster, and monitors the availability of the other nodes by exchanging heartbeats with its neighboring nodes. If a node stops sending heartbeats, the surviving Cluster Managers on the remote nodes in the cluster take the necessary actions to get the critical applications up and running and to ensure that data has not been corrupted or lost. The available Cluster Managers take over the network interface, the volume groups or disk drives, and then restart the applications.

The Cluster Manager monitors local hardware and software subsystems, tracks the availability of other nodes in the cluster, and monitors the availability of the other nodes.

As part of the takeover, the Cluster Managers delete and re-create their routes, and refresh the Address Resolution Protocol (ARP) cache of any clients. This allows clients to connect to the backup node using the same address originally used to connect to the primary node. Any client transaction in mid-flight during a node failure must be resubmitted. This does not cause data corruption since the in-flight transaction has not yet been committed.

The takeover time varies depending on the amount of resources that need to be acquired by the takeover node and the amount of application recovery processing required. In most configurations, node takeover completes within one to five minutes.

Effect on Client Applications

Node failure causes most naive client applications to hang, since the connection between the client and server is broken. After node takeover has completed, the system administrator must notify the users, who then must log back in, restart their applications, and retrace their steps to get back to where they were before the node failure occurred.

Intelligent applications, on the other hand, can maintain the application state until the server is reconfigured, and can inform users of the nature of the problem and when it should be resolved. Intelligent applications can take action; for example, they could automatically switch to a new server in case of any fallover, making the change transparent to users.

Making an Application Cluster-Aware

Application developers can use the Clinfo libraries to develop sophisticated applications. For example, the cluster status monitor included with the HACMP software was written with these libraries. Our intent in this article, however, is to demonstrate how easy it is to use the Clinfo libraries to write simple yet powerful cluster-monitoring facilities. We now discuss two C language routines, `hacmpStable` and `checkConnection`, that enable a front-end application to handle node failure while maintaining its connection to a Sybase SQL Server database.

The hacmpStable Routine

The `hacmpStable` routine queries an HACMP cluster for its status (availability for processing) and reports the status to a calling procedure (in this case, the `checkConnection` routine, described below). The status of an HACMP cluster is determined by both its state and substate. A cluster can be in one of three states:

- `CLS_UP`: At least one node in the cluster is up, and a primary is defined.
- `CLS_DOWN`: At least one node in the cluster is up, but a primary is not yet defined.
- `CLS_UNKNOWN`: The Cluster SMUX Peer daemon cannot communicate, or is not yet communicating with, an active Cluster Manager daemon.

In addition to its state, a cluster can be in one of the following substates:

- `CLSS_STABLE`: The cluster is stable (no reconfiguration is occurring).
- `CLSS_UNSTABLE`: The cluster is unstable (a change in topology is occurring).
- `CLSS_ERROR`: A script has failed; the cluster has been in the process of configuration (unstable) for too long.
- `CLSS_UNKNOWN`: The Cluster SMUX Peer daemon cannot communicate, or is not yet communicating with, an active Cluster Manager daemon.

An application running on a client machine can only communicate with a server running on a cluster node when the cluster's state is up and its substate is stable.

```
struct cl_cluster {
     int  clc_clusterid;                /* cluster id       */
     enum cls_substate clc_substate;   /* cluster substate  */
     enum cls_state clc_state;          /* cluster state     */
     char clc_primary[CL_MAXNAMELEN]; /* primary node name */
     char clc_name [CL_MAXNAMELEN];    /* cluster name      */
};
```

Figure 2. The cl_struct data structure

```
/*
 *  Routine that uses Clinfo API calls to determine
 *  the state and substate of an HACMP cluster
 */

int
hacmpStable(int clusterId)
{
   static int           firstTime = 0;

   int                   result;
   struct cl_cluster    clstr_buf;

   if (firstTime) {
      /*
       * Create initial Clinfo connection and allocate buffer arrays.
       */
      result = cl_initialize();
      if (result != CLE_OK) {
         (void)fprintf(stderr, "cl_initialize failed, %s\n",cl_errmsg(result));
         return TRUE;
      }
      firstTime++;
   }

   result = cl_getcluster(clusterId,&clstr_buf);
      if (result == CLE_OK) {
         if (clstr_buf.clc_state == CLS_UP) {
            if (clstr_buf.clc_substate == CLSS_STABLE) {
               return TRUE;
            }
         }
   } else {
      (void) fprintf(stderr, "cl_getcluster(%d) failed, %s\n",
                     clusterId, cl_errmsg(result));
   }
   return FALSE;
}
```

Figure 3. The hacmpStable source code

The hacmpStable routine uses three functions from the Clinfo C library:

- The cl_initialize routine checks to see if Clinfo is running and, if so, acquires the shared memory map. This map is managed by Clinfo using information stored in the MIB by the Cluster SMUX Peer daemon.
- The cl_getcluster routine returns information about a specified cluster in a cl_cluster data structure. The cl_cluster data structure is shown in Figure 2.
- The cl_errmsg routine takes a status code returned by Clinfo and returns the text associated with that status code.

The source code for the hacmpStable routine is listed in Figure 3.

The checkConnection Routine

The checkConnection routine determines if a client's connection to a database is still valid. If the connection is good, the checkConnection routine returns success to the calling procedure, which then continues processing. If the connection is no longer valid (for example, the node has failed), the checkConnection routine calls the hacmpStable routine to determine if the cluster is stable. If the cluster is not stable, the checkConnection routine loops, waiting for the cluster to become stable. When the cluster stabilizes, indicating that the takeover node has acquired all the resources from the failed node, the checkConnection routine then reestablishes the client application's connection to the database. The client application then resumes processing.

```
/*
 *  Routine that tests a database connection to determine
 *  if it is valid. If not, calls the hacmpStable routine to
 *  determine cluster status.
 */

int
checkConnection()
{
   char *clusterStr;

   clusterStr = getenv("CLUSTERID=");
      if (clusterStr) {
         if (!DBproc || (DBDEAD(DBproc))) {
            while (1) {
               int clusterId = atoi(clusterStr);
               if (hacmpStable(clusterId)) {
                  (*msgCallback)("Cluster stable, attempting to reconnect");
                  (void)initConnection(0);
                   break;
               }
               (*msgCallback)("Cluster unstable, waiting...");
               sleep(5);
            }
            return 0;
         }
   }
   return 0;
}
```

Figure 4. The checkConnection source code

To enable a front-end application to handle node failure transparently, an application developer would place the `checkConnection` routine before every SQL execution. The client application is then able to maintain its connection to the database server, and reconnect only if necessary.

The source code for the `checkConnection` routine is listed in Figure 4. Note that this routine was written using the Sybase SQL Server DBlib library.

Using Clinfo Routines in Client Applications

To use the `hacmpStable` and `checkConnection` routines in client applications, application developers must be sure to include the proper header files and link to the necessary library.

Header Files

Application developers must specify the following `include` directives in each source module that uses the Clinfo C library:

```
#include <sys/types.h>
#include <netinet/in.h>
#include <cluster/clinfo.h>
```

Linking the libcl.a Library

Application developers must add the following directives to the object load command of a single-threaded application that uses the Clinfo C library:

```
-lcl  -lclstr
```

The `libcl.a` library contains the routines that support the single-threaded Clinfo C library. The `libclstr.a` library holds commands and other information that may be used by some of the `libcl.a` routines.

Steven Kohler, CLAM Associates, Inc. 101 Main Street, Cambridge, MA 02142. Internet: slk@clam.com. Mr. Kohler is principal engineer in CLAM's technical support group, and is the technical lead for the HACMP for AIX change team.

Thomas Casey, CLAM Associates, Inc. 101 Main Street, Cambridge, MA 02142. Internet: tom@clam.com. Mr. Casey is the manager of CLAM's technical writing group. He has a BA from Trinity College in Hartford, Connecticut and an MA from Emerson College in Boston.