SKY's Health Maintenance Strategy Yields High Application Availability

Gerry Pocock, Ph.D.

SKY Computers Inc.

1 Introduction

There are two main methods that can be employed to build a system that provides High Application Availability (HAA). The first is to construct components, both hardware and software, that exhibit a very long mean time between failures (MTBF). The longer this time, the better the system is at supporting HAA. Systems that exhibit high MTBF are very expensive to build, so much so, that their cost can be prohibitive. For example, one can build a system using triple redundancy. If the system's baseline processing requirements called for several hundred processors, tripling the number of processors in order to provide a high MTBF would probably make the system too costly.

The second method that can be employed to build a HAA system is to construct the system such that the mean time to repair (MTTR), i.e., the length of time it takes to repair a fault, is very short. This method will not work for critical applications which cannot tolerate any down time. On the other hand, for those applications in which a minimal amount of down time is acceptable, reducing MTTR is an acceptable method of producing HAA systems. Indeed, combining both MTBF and MTTR techniques can produce a very cost effective HAA system.

This combination of using both MTBF and MTTR techniques is the approach taken by SKY for its newest generation of products. High quality hardware and software components are being used to build the new systems. These quality components will product high MTBF times. In order to achieve low MTTR times, SKY is providing with its new product line a Health Maintenance System (HMS), composed of both hardware and software, that will help its customers and their applications maintain the health of the system in order to product low MTTR times.

We have chosen to name the new system the Health Maintenance System because it provides services similar to those provided by health maintenance organizations. It allows the customer and their applications to determine the health of the system (symptoms), to take actions to deal with system problems as they arise (medicine, surgery, etc.), and to pro-actively uncover potential problems (check up, stress tests, scans, etc.).

The basic principles of the Health Maintenance System are similar to principles underlying Recovery Oriented Computing (roc.cs.berkeley.edu). ROC is based on the premise that hardware,

software, and humans errors are not problems to be solved, but rather facts that have to be lived with, and that fast recovery is the right way to live with these facts. Even though ROC is targeted at WEB based applications and HMS is targeted at HPEC applications, they both attempt to increase application availability by decreasing MTTR.

This paper presents the software architecture of the HMS framework. The fundamental building blocks of this framework are the resources and the managers that are responsible for managing the resources. Resources are the fundamental components, hardware and software, of the system. The HMS components are similar, but not identical, to the entities as specified in the Service Availability Forum's (SAF) Hardware Platform Interface (HPI) (www.saforum.org). The managers provide access to these resources such that users and their application can monitor and control the resource. The sections that follow will provide more specific information about resources and managers, and how applications interact with the managers.

2 Resources and Resource Managers

Simply put, a resource is a fundamental component of the system that an application (or user) needs to manage in order to maintain the overall health of the system. When we talk about managing a resource, we are referring to three primary activities:

1. Sensing the state of the resource, e.g., is it functioning correctly.
2. Performing some operation on the resource in order to improve its health, e.g., increasing the speed of a processors fan because the processor's temperature is too high,
3. Performing diagnostic operations to determine if their exists any latent problem with the resource, e.g., stressing an interconnect.

It is through a combination of these three activities that the health of a system can be maintained. It is the responsibility of the resource manager to provide these three capabilities. Before we look in detail at how this is achieved, we first discuss the primary resources that are supported by the Health Maintenance System.

Many HAA frameworks, e.g., IPMI and the SAF HPI, have chosen to focus solely on hardware. In these frameworks, components or entities are strictly hardware, generally field replaceable units, e.g., blades. In HMS, we have extended this model of resource to include software as well as hardware. For example, the processing node resource is a combination of hardware, processor and memory, as well as the operating system. Thus, for a processing node, one can not only determine the type of processor, e.g., a PowerPC, by also its load (what percentage of the CPU cycles were spent executing code as opposed to being idle).

Another extension to the basic resource model that is supported by the HMS system is the container resource. In addition to having a set of attributes, e.g., temperature, container resources contain other resources. For example, a chassis resource contains board resources which themselves contain processing node resources. This extension allows the HMS system to model a system as a hierarchy

of resources, starting with the *system* resource and stopping at the base component resources. Thus, checking the health of the entire system is as easy as checking the health of the system resource.

The following is a listing of the different classes of resources that are supported by the default HMS system. As will be discussed later, this list can be easily modified to support a particular application.

- Processing Node Resource: Resources of this type are the system's basic processing components. They are generally composed of some number of processors, memory, and interconnect bridges. In addition, they may also contain data transfer engines.
- Device Node Resources: Resources of this type are the system's additional support components. Examples include I/O devices and disc systems.
- Interconnects: Resources of this type provide are the system's data exchange components. Examples including PCI and Infiniband interconnects.
- Interconnect Bridge Resources: Resources of this type are the system's bridges that transfer data from one interconnect to another. Examples include the HCA.
- Electrical Resources: Resources of this type are the system's electrical components. Examples include power supplies and converters.
- Environmental Resources: Resources of this type are the system's environmental support components. Examples include fans.
- Board Resources: Resources of this type are the system's boards. They include blades as well as baseboards and backplanes. Board resources are containers that can contain any number of component level resources such as the processing node resources.
- Chassis Resources: Resources of this type are the system's chassis. Chassis resources are containers that contain boards as well as other component level resources such as power converters.
- Rack Resources: Resources of this type are the system's racks. Rack resources are containers that contain chassis as well as other component level resources such as power supplies.
- System Resources: Resource of this type are the entire system.

Resources are an abstraction that are not realized by the Health Maintenance System. In other words, the system does not support any notion of a software resource construct. Instead, HMS embeds the resource abstraction within the resource manager abstraction, and it is the resource manager abstraction that is realized. In other words, for every resource in the system, their exists a resource manager, a software construct, that is responsible for providing the fundamental support such that an application can manage the resource (see the three primary management activities listed above).

For each resource class listed above, a virtual resource manager class is defined. A specific resource manager class is then derived from the corresponding virtual resource manager class. It is from these classes that specific resource manager instances are instantiated.

Each manager class will provide a set of methods that, at a minimum, provide the following functionality:

1. Query methods: These method provide support for querying the resources static, e.g., manufacture ID, and dynamic, e.g., the resource's state, information.
2. Management methods: These methods provide support for managing the resource including directly changing the state of the resource, e.g., changing the speed of a fan, as well as setting the manager's management policies, e.g., making the manager issues a notification when the processing node's temperature exceeds some bounds.
3. Testing methods: These methods provide support for testing the resource. These testing methods provide the capability for pro-actively finding potential problems with the resource and can range from very simple methods, e.g., ping a node, to more complex methods, stress testing the InfiniBand Fabric. In addition to supporting a range of self-contained tests, the manager must also support tests that work in conjunction with other tests. For example, a board resource manager will rely upon processing node resource managers to test an interconnect fabric that connects the various processing node resources.

Resource mangers will be realized as CORBA objects.

3 HMS Architecture

Before we provide more details concerning the resources and the resource managers, we first present the architecture of the Health Maintenance System. As shown in 1,

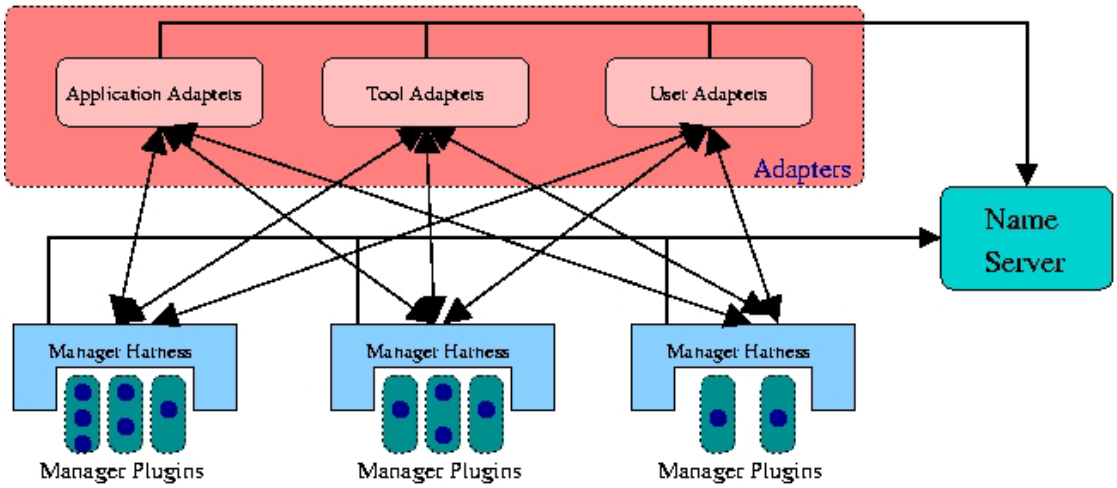

Figure 1: The Overall HAA Architecture

the system is essentially a hierarchy composed of three layers.The top layers consists of a set of adapters which provide an appropriate interface via which clients can manage the system resources. These adapters are discussed in more detail in section 3.1.

The middle layer of the architecture consists of the various Node Harnesses. These harnesses, which are discussed in more detail in section 3.2, primarily server as a transport mechanism between the various adapters and the resource managers. However, the harnesses also provide some common functional support for the resource managers.

The lowest layer of the architecture consists of the resource managers. These managers are packaged into containers called manager plugins that can be plugged into a manager harness. The plugins can contain any number of resource managers. The resource managers are represented in the figure as the dark blue circles.

In addition to these three layers, the architecture also includes a name services component which is responsible for providing a mechanism by which an interface can determine which manager is responsible for a specific resource. The name services is discussed in section 3.4.

3.1 Adapters

Clients that will use the HMS middleware to control resources fall into one of three classes. These are:

1. applications,
2. system management tools, and
3. system administrators.

The Adapter layer of the HMS middleware has been architecture to support these three different classes of clients by providing adapters that support appropriate interfaces that can be used by these clients. The adapters fall into three corresponding classes:

1. An Application Programming Interface (API) adapter provides the interface, the API, that a running application can use to manage the resources. An example of such an adapter is one that supports the Service Availability Forum's (SAF) Hardware Platform Interface (HPI).
2. A Tool adapter provides the interface that a system management tool can use to manage the resources. An example of such an adapter is a Simple Network Management Protocol (SNMP) agent.
3. A User interface provides the interface that a system manager can use to manage the resources. An example of such an adapter is a WEB server, e.g, the Apache WEB server, coupled with the appropriate CGI executables.

All adapters work in a similar manner, i.e., they translate requests made by a client using the interface they provide, into requests that the HMS middleware can handle. The architecture of the HMS middleware places no restriction on the interface that is supported by the adapters. These interface can include standard function invocations as well as messages passed by standard sockets.

The architecture does specify that CORBA will be used for the internal data transport and method invocation. The IDL used internally by the HMS middleware should be based on the Data Distribution Service IDL that is current under consideration by the OMG.

3.2 Manager Harness

A Manager Harness has two primary responsibilities. It provides a run-time environment for a resource manager and it provide the underlying transport that allows the resource managers and the adapters to communicate.

Upon startup, a harness will load a predefined set of manager plugins and activate each manager contained in the plugin (see section 3.3 for a description of how this is accomplished). The harness should be constructed such that it is possible to dynamically load and unload resource managers, i.e., the plugins, at any time.

The harness must provide the require call backs so that resource manager can register itself and the resource it manages with the harness. It is the responsibility of the harness to register this information with the Name Server. In this way, adapters will be able to determine which resources are in the system and which manage is responsible for each resource.

A Manager Harness must register itself with the Name Server so that the adapters can directly access the harness, i.e., query it and invoke operations. The supported operations must include (at a minimum):

- the ability to suspend and resume a resource manager
- the ability to kill and unload a resource manager
- the ability to load and start a resource manager
- the ability to kill the harness
- the ability to query the harness' configuration
- the ability to query the harness' state
- the ability to query the identification of the resource managers running in the harness
- the ability to query the identification of the resources managed by the managers running in the harness
- the ability to invoke an operation on any specified set of resource managers running in the harness
- the ability to invoke an operation on any specified set of resources managed by the managers running in the harness.

In addition to these operations, a harness should also support operations that are commonly used by a significant number of resource managers. For example, many managers will periodically sample the state of any attribute in order to detect whether the state of the attribute has gone out of bounds. Providing a mechanism within the harness that can be used by the resource manager to periodically sample the attribute's state will be generally useful.

The architecture doesn't specify how many manager harnesses are required by the system. However, generally there will be one such harness running on every node in the system.

3.3 Manager Plugins

Manager plugins are basically a container for resource managers. The plugin is constructed as a dynamically loadable library that contains a set of resource managers. Once constructed, the plugin can be dynamically loaded by the harness using `dlopen()`. Once loaded, plugin's entry point is invoked resulting in the creation of the resource manages contained within the plugin.

As discussed in section 2, resource managers are the software components that are responsible for managing a resource. A manager may provide some level of local control of the resource. Such local control will include taking some predefined action should the resource deviate from the norm. Managers that provide such local control will generally be constructed such that the action that is taken and the value used to determine if the resource has deviated from the norm will be configurable. Such configuration can be done either at the time the manager starts (built-in defaults) and/or during run-time (via a request from a client).

In addition to the local control, a manager also provides support for global control. This control takes the form of handling requests from a clients. Such requests will include:

1. requests for information regarding the resource it is managing, e.g., the name of the resource, its manufacturer's ID, etc., , and
2. requests for some operation to be performed, e.g., getting the attributes current state of the resource, resetting the resource, etc.

A resource manager is free to support other types of requests that are directly pertinent to the resource being supported.

Resource managers are to be constructed such that they execute within the context of the manager harness and use the support functionality provided by that harness, e.g., the harness' communication transport mechanism. In order to achieve this, resource managers are combined into a single loadable module, e.g., a dynamic loadable library, called a plugin that can be loaded into the harness' run-time context.

3.4 Name Server

The name server is responsible for providing a simple name service. The name service includes the ability to look up the name of every component in the system including the Manager Hardnesses and the Resource Managers. Associated with each component name is a handle to the component via which that component can be accessed. Examples of such handles would include the triplet of IP address, port number, and component name that can be used with standard sockets to provide communication between the adapter and the component, as well as a CORBA reference.

Given that the name server is the mechanism used to find all other components in the system, a systematic mechanism must be established that will allow both manager harnesses and adapters to obtain an initial reference to the name server. Common mechanism used for this purpose include the use of broadcast or multicast to establish the initial communication as well as global files that are used to hold a reference to the name server.

4 Summary

The Health Maintenance System (HMS) is to be a major component of SKY's High Application Availability (HAA) strategy. It will aid applications in maintaining a low mean time to repair (MTTR) that will enable these applications to achieve a high degree of availability.

Gerry Pocock 2004-03-08