

A new Cluster Resource Manager for heartbeat

Lars Marowsky-Brée
lmb@suse.de

January 2004

1 Overview

This paper outlines the key features and design of a clustered resource manager to be running on top of and enhancing the Open Clustering Framework infrastructure provided by heartbeat.

The goal is to allow flexible resource allocation and globally ordered recovery actions in a cluster of N nodes and dynamic reallocation of resources in case of failures (*fail-over*) or in response to administrative changes to the cluster (*switch-over*).

This new Cluster Resource Manager is intended to be a replacement for the currently used resource manager of heartbeat. However, due to compatibility reasons, heartbeat will continue to support both for a long time and allow the administrator to select the preferred one.

1.1 Requirements overview

The CRM needs to meet at least the following requirements:

- *Secure and simple.*

heartbeat already provides these two properties very well. For sanity and stability – both of the developers, but in particular the users –, complexity needs to be kept at a minimum (but no simpler). Right now, for example, the configuration files need to be manually synchronized among the cluster nodes. This is acceptable for two nodes – but obviously, people already get it wrong all the time –, but not particularly scalable.

Security is an important requirement; building on top of the heartbeat infrastructure provides us with a lot of shielding against network attacks, but care must be taken while implementing and designing the system.

- *Support for more than 2 nodes.*

The current RM supports only two nodes. This is clearly no longer sufficient for bigger clusters in data centers or built from blade systems.

- *Fault tolerant.*

The CRM must be able to cope with node failures, network failures such as full link failures or the cluster becoming partitioned, but also with failures of individual resources. The current RM does not monitor resources themselves, which is the most distinct missing feature.

- *More complex policies.*

The current RM only supports very simple resource group based policies, where a resource group either runs on one of the two nodes or it does not.

Scaling up, this is no longer sufficient. A production system may “push off” the test system from the nodes because it has higher priority and requires exclusive use of the node. Native support for replicated resources with N masters and M slaves is also a key feature, as is support for resources which can be instantiated on more than one node.

- *Administrator feedback.*

The system must be able to provide the administrator with an up-to-date status of the cluster at all times, and also respond immediately to administrative requests for migrating resources et al. These feedback chains should also be exploitable to facilitate the building of CIM or SNMP frontends.

- *Extensible framework.*

As far as it is possible and sensible, the framework should not impose limitations and at the same time separate the different components so that each module itself is of a manageable size.

1.2 Scenario description

The design outlined in this document is aimed at a cluster with the following properties:

- *Cluster membership service* The cluster provides a Consensus Membership Layer, as outlined in the OCF documents and provided by the CCM implementation by Ram Pai.

This provides all nodes in a partition with a common and agreed upon view of cluster membership, which computes the largest set of fully connected nodes¹.

It is possible for the nodes in a given partition to order all nodes in the partition in the same way, without the need for a distributed decision. This can be either achieved by having the membership be returned in the same order on all nodes (as CCM does), or by attaching a distinguishing attribute to each node.

¹As this is a NP-complete problem, heuristics are used.

- *Cluster messaging service*

The cluster provides a communication mechanism for unicast as well as broadcast messaging.

Messages do not necessarily satisfy any strong ordering requirement such as totally ordered or causal ordering, but they are atomic. The CRM should not require a complex group messaging service, as they are simply not yet available on Linux in a well-defined form.

- *Failures are relatively rare.*

Of course, failures do occur, otherwise we would not need to have this software in the first place; however, the key point here is that they are rare, and that we need to run with as little impact as possible while no failures are present.

As long as the system reacts to a failure with minimal latency, we are fine. Typical fail-over times are lower-bound by the time of the resources starting up, not by the cluster manager reaction. We can initially trade a little bit of efficiency for simplicity.

Byzantine failures in particular are very rare and do not need to be dealt with efficiently; these errors are self-contained to the respective components which take appropriate precautions to prevent them from propagating upwards. (Checksums, authentication, error recovery or fail-fast systems.)

Stable storage is stable and not otherwise corrupted; network packets are not spuriously generated et cetera. (The heartbeat messaging layer protects against these network failures.)

- *Time synchronization.*

Time is synchronized cluster-wide. Even in the face of a network partition, an upper bound for time diversion can be safely assumed. This can be virtually guaranteed by running NTP across all nodes in the cluster.

This is not a strong requirement as we take measures to protect against time drifts (such as generation counters), but simplifies a few assumptions and will generally make the life of the administrator more enjoyable when reading logfiles.

- *IO fencing is available.*

IO fencing allows us to prevent a failed node from modifying shared resources such as disks. Whether or not IO fencing is needed of course depends on the resources present in the cluster, however the assumption is that it is commonly required and thus needs to be well supported.

The fencing mechanism provides definitive feedback on whether a given fencing request succeeded or not.

Additionally, we aim to support not only node-level granularity (such as provided by the STONITH mechanisms), but also support resource-level fencing (such as SCSI reservations).

- *A node is authoritative for its own status.*

A node knows which resources it currently holds and their state (running / failed / stopped); it can provide this information if queried and will inform the CRM of state changes. Thus ultimately, the status of the whole cluster can be recovered by querying all nodes, avoiding the need for distributed book keeping.

This part will be provided by the Local Resource Manager, which is not strictly within the scope of this paper. However, it will be outlined below to provide the reader with the full picture.

2 Design overview

2.1 Components

To meet these requirements, the system is divided into several components as follows:

- *Cluster infrastructure* The cluster infrastructure is provided to us by heartbeat itself. We use its core features and also some libraries, but the most important features for this discussion are:
 - heartbeating and node and link failure detection.
 - Cluster-wide messaging.
 - Consensus Cluster Membership.

For details regarding heartbeat, please see Alan Robertson's talk at this conference.

- *Local Resource Manager* The LRM is responsible for managing and tracking the resources on the nodes. Resources are the typical resources encapsulating an application software, IP address or a filesystem mount, but also include the fencing devices. In short, the LRM is doing all the actual work.
 - LRM itself
 - Executioner
 - Resource Agents
- *Cluster Resource Manager* Last but not least, the CRM itself coordinates all these across the cluster. This is the topic of this paper and will be outlined in more detail below.
 - Cluster Information Base
 - Policy Engine
 - Transitioner

2.2 Mode of operation

The basic doctrine is to simplify everything as far as possible. Obviously, running in a distributed system, we do have some complexity we cannot avoid, but we try to avoid distributed decision processes or at least encapsulate them, or preferably build upon the distributed decision already made by a layer below us.

The Concensus Membership layer already provides us with an ordered list of nodes. Using this, we elect a single **Designated Coordinator** for the cluster which then takes over all management functionality. All events (node failures, resource failures, but also admin requests) are processed by it and all responsive actions are triggered from here.

Whenever an event is received by the DC, the current cluster configuration plus the current cluster status are retrieved and also broadcast to all alive nodes. Based on this, the policy engine computes a transition graph, an ordered dependency graph of all recovery actions necessary to restore the cluster again, which is then executed.

Executing the transition graph amounts to a full graph traversal; each node in the graph is an action to be performed by a participating LRM, such as (re-)starting or stopping a resource or STONITHing a node. A new node (action) can be begun as soon as all actions it depends on have successfully completed; this allows multiple operation to be performed in parallel.

If any new event occurs during this graph traversal, the graph traversal is stopped after all currently in-flight operations have finished. Then the algorithm is invoked again, and a new transition graph computed.

2.3 Dependency based resource configuration

Many currently available systems still employ a resource-group based model. In this model, the resources are grouped into (surprise!) **Resource Groups**, which are handled as the atomic unit for fail-over or switch-over. A resource group typically is associated with a list of preferred nodes on which it should run; within the resource group, there is a static start/stop ordering between the resources.

However, this model does not adapt very well to more complex scenarios. Replicated resources, resources which are active on multiple nodes at the same time, negated relationships between resources are hard to express cleanly this way.

The new model, which is already used by other projects and products too, is resource dependency based. Each resource has dependencies of various kinds on other resources or locations. For example, the minimal dependency required to model the existing resource group approach is *started after resource X on the same node*.

Other dependency types include *not on the same node as, started after, but regardless of node placement, preferably not on the same node as* et cetera.

This model also allows us to restrict failures to only the critical path, without affecting the rest of the resources. Another major feature is the ability to parallelize resource operations as far as possible, without the serialization inherent in resource groups.

It is obvious that the new model provides a superset of the old capabilities; a loss-free

automated conversion to the new scheme is thus theoretically² possible.

The configuration becomes more graph oriented; this is a paradigm change compared to resource groups, but at the same time more natural.

3 The Cluster Resource Manager

The CRM itself breaks down further into several components.

3.1 The CRM core

The core is mostly a message switch.

It is also responsible for the leader-election in the cluster. The coordinator is a “primus inter pares”; in theory, any CRM can act in this fashion, but the arbitration algorithm will distinguish a designated node.

On the Designated Coordinator, it retrieves the current status from the LRMs on all nodes and consolidates them with the current cluster configuration; this information is then handed to the Policy Engine and then on to the Transitioner.

On the regular cluster nodes the CRM relays the commands to the LRM and feeds the events back to the DC. (Events are either reported by the LRM or by administrative clients.)

3.2 Cluster Information Base

The CIB holds the configuration in a simple XML tree, versioned with a timestamp and generation counter. Using this information, the most recent coherent configuration available in the cluster is identified by the DC and then updated on all other nodes.

This ensures that the cluster is always working from the most recent configuration and avoids inconsistencies.

Any updates to the configuration will be serialized by the DC; they will be verified, committed to its own CIB and also broadcast to all nodes in the partition as an incremental update.

Configuration data in the CIB includes:

- Configured resources
 - Resource identifier
 - Resource instance parameters
- Special node attributes
 - Attachments to specific storage subsystems
 - Architecture flags
 - Possibly auto-discovered

²If someone writes the code.

- Administrative policies
 - Resource placement constraints
 - Resource dependencies

3.3 Policy Engine

The policy engine receives the current cluster state and configuration from the CRM and computes the transition graph.

This includes figuring out which resources have to be stopped and where they are moved to; which are started up from scratch; which cannot be started in the current state because nodes are unavailable; which implicit dependencies such as IO fencing have to be performed et cetera.

Clearly, this component includes the most complex algorithms of the whole. This is why it is shielded as far as possible from the details of cluster messaging and remote command execution, which it does not have to deal with.

At the same time, centralizing this computation is orders of magnitude simpler than trying to perform it in a distributed fashion.

3.4 Transitioner

The Transitioner encapsulates the execution of the transition graph. It instructs the LRMs in the cluster and waits for their positive replies to its commands.

Whenever a negative reply is received, all currently in-flight operations are allowed to finish and the graph aborted at this barrier. The CRM will then provide this state to the Policy Engine once more, until eventually the cluster converges.

If the graph traversal has run to completion successfully, the CRM becomes idle, waiting for new events to occur.

4 Local Resource Manager

Note: This section only documents the requirements on the LRM from the point of view of the cluster-wide resource management. For a more detailed explanation, see the documentation by Alan Robertson et al on the Linux HA Wiki..

This component knows which resources the node currently holds, their status (running, running/failed, stopping, stopped, stopped/failed, etc) and can provide this information to the CRM. It can start, restart and stop resources on demand. It can provide the CRM with a list of supported resource types.

It will initiate any recovery action which is applicable and limited to the local node and escalate all other events to the CRM.

Any correct node in the cluster is running this service. Any node which fails to run this service will be evicted and fenced.

It has access to the CIB for the resource parameters when starting a resource.

NOTE: Because it might be necessary for the success of the monitoring operation that it is invoked with the same instance parameters as the resource was started with, it needs to keep a copy of that data, because the CIB might change at runtime.

5 Further reading

In search of clusters by Gregory F. Pfister

Distributed Algorithms by Nancy Lynch

Fault Tolerance in Distributed Systems by Pankaj Jalote

Transaction Processing: Concepts and Techniques by Jim Gray, Andreas Reuter.

Blueprints for High Availability by Evan Marcus, Hal Stern