Reprinted from the

Proceedings of the Ottawa Linux Symposium

June 26th–29th, 2002
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
Stephanie Donovan, *Linux Symposium*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Wild Open Source, Inc.*

The Open Clustering Framework

Lars Marowsky-Brée
Research & Development
SuSE Linux AG
lmb@suse.de

Abstract

The Open Clustering Framework is an effort to unify the current projects underway with regard to clustering on the Linux platform with a common component model, common APIs and at least one Open Source implementation. The initial focus is on High Availability Clustering and Linux, but this is not a long-term or inherent limitation.

1 Introduction

1.1 Current status of clustering on Linux

Many, many High-Availability clustering projects have appeared on Linux in the last few years; not only Open Source ones, but also almost every major vendor has ported their solution from their proprietary operating system to Linux.

Today, we have at least ten Open Source clustering solutions for Linux, and in excess of *twenty-five* commercial ones. A solution exists for almost every niche. In fact, it is safe to say that at least *N+1* solutions exist for every niche. For an overview, see the very useful *Linux Clustering Information Center*[1] by Joe Greenseid.

And this number is still constantly growing.

Linux enjoys an abundance of riches. Users, application or operating system programmers and system-integrators have many choices available for building the perfect solution from these many pieces.

This sounds like the topic can be safely considered solved...

1.2 The problem

Unfortunately, the pieces do not fit together **at all**.

They share neither a common API, nor a common concept of what (High-Availability) clustering is. This may seem surprising at first, because at first glance most of them appear to solve the same general problems and provide mostly the same functionality; largely, they are even structured roughly in the same way.

This is actually true; and the reasons why they do not fit together are largely historic. As already said, many of them were written inde-

[1] http://www.lcic.org/

pendent of each other, be it due to commercial interests, the *I just want to solve this part*-phenomenon, *My thesis has to be academically perfect*, plain ignorance or any other number of reasons.

On the proprietary platforms, there is the nine-hundred pound gorilla[2] called *The Vendor* defining the standard; and maybe one or two competing products, which would either do something fundamentally different or complement the vendor's solution.

On Linux, such a single vendor does not exist[3] for obvious reasons, which is commonly considered a good thing. It also implies that no-one has a higher right to claim they are the standard than the rest, unless they would be both vastly superior and captured a substantial share of the market.

However, the Linux clustering *market* is sufficiently evenly fragmented that this is true of no-one; everybody can claim roughly the same weight as the rest, so no API, no conceptual model has established itself as a reference.

1.3 Consequences

All solutions are incompatible with the rest; they will not inter-operate. And what is worse, even if they do *appear* to inter-operate successfully, it is by pure chance; not the best foundation for a highly-available mission-critical system and bound by Murphy to fail at the most inconvenient time.

They have different APIs, different concepts of what a *cluster* is, slightly different approaches of how to cope with specific failure situations and so duplicate large parts of the picture between them. This can potentially lead to all sorts of nasty deadlocks, data corruption and loss of service.

[2] Metaphor courtesy of Alan

[3] Despite every Linux vendor claiming it is them.

1.4 Affected parties

Everyone has a different perspective on the problem; but the short summary is: **Everybody loses**.

1.4.1 The end-user

The end-user usually could not care less about how his problem is solved; he will just care whether it is solved, and what it costs. He is not particularly amused that yes, almost all his problems can be solved, but that a coherent solution is not possible; data has to be replicated, independent systems running different parts of the solution have to be purchased, installed and maintained, and that he has to spend an amazingly large sum on making it all work.

A special case of the end-user is the system administrator[4]; he has to maintain the whole mess of different solutions with different interfaces, and not even a common *SNMP MIB* for monitoring exists for the most basic aspects of the whole.

1.4.2 Independent software vendors

Independent software vendors want to sell their application to the largest audience possible; preferably, everyone. Having a very heterogeneous environment means to either abandon a large part of the market, cope with many different solutions or implement the entire stack

[4] Put away the LART, will you.

themselves and be independent of the mess[5].

Each of these approaches has its own problems; be it a smaller potential market, huge compatibility matrices or lots of code which has nothing to do with the core competency of the company.

1.4.3 Commercial providers of clustering solutions

A provider of a clustering solution has largely the same problems as the independent software vendors; they do not have much of a choice with regard to the scope of their solution. Even if they just want to solve part of the problem—for example *just* a cluster-aware filesystem—they are either stuck with building the full package or tieing themselves to one or more other vendors.

Even vendors of full solution stacks aren't better off by far; they have to compete fiercely for the attention of the end-users and the independent software vendors to support *their* version of the wheel. If they are just slightly better for a particular niche which would theoretically be enough for them to make do or even be off pretty well, it may not be large enough for an ISV to add compatibility for their solution.

1.4.4 Open Source projects

Not only does everything said before apply here; but the Open Source community faces a rather severe additional problem which is often overlooked: Split of rare resources.

While it can be generally considered a good thing to have two or even more solutions for a particular area to keep competition up, being split between so many different projects prevents many of them from reaching critical mass and becoming really successful.

This is especially true of an area which is inherently complex, has a limited audience or simply requires more resources to work on than the common programmer has at home, like multiple nodes for a cluster.

1.4.5 System integrators

System integrators are in a tricky position; all choices what to include are wrong. They are unable to provide a coherent clustering solution without annoying the other ninety-five percent of the market.

1.4.6 Consulting companies

If the solution is build by a consulting company, one might assume they are in heaven – after all, lots of complexity means longer projects, which means more money. Or, as *despair.com* puts it: *If you are not part of the solution, there is good money to be made in prolonging the problem*.

This is a false guess; consulting companies prefer to be paid for not doing any *real work*, and this is largely inevitable if you want to deliver a working clustering solution on Linux because of the above factors.[6]

[5] Which is what Oracle 9i Real Application Cluster does.

[6] You did notice the tongue-in-cheek, didn't you.

2 The solution

After hopefully having demonstrated that everyone will benefit from having this problem solved, let's look at the two obvious solutions in more detail:

2.1 Copy-cat standard

The first thoughts were to look at other platforms; was there a standard which could easily be adopted – POSIX, *best current practices* or even a de facto industry standard.

The search was unsuccessful; neither a coherent full standard exists nor a subsets of the whole[7]. All other platforms have said nine hundred pound gorilla setting the standard by vendor decree. POSIX has not yet bothered to work on this topic, nor has any other larger standards body.

Mimicking one of the vendor standards from another platforms also has intellectual-property issues and the problem that you cannot get any other commercial vendor to agree to it.

2.2 Standard by appointment

The next obvious solution to the problem is to do as the other platforms: just pick one solution and declare it the standard; many affected parties will not care how the standard appeared, as long as it makes their pain go away.

This has been tried by every vendor and even many Open Source projects; the issue is that no-one so far had sufficient weight for it, and those who do had political reasons not to.

Another issue comes with the fact that none of the solutions solves all aspects; a complete, coherent solution does not exist, despite all marketing claims.

Having more than two or three solutions be *the standard* does not solve the problem; so unless a solution which is agreeable for everyone or at least a large enough part appears, this does not work on Linux; compare the lack of a nine-hundred pound gorilla described before.

3 The framework

3.1 Mission statement

And this is where the *Open Clustering Framework* enters the picture; the easier routes have been considered, but found to be unusable. Everybody sighed, shrugged and figured that real, not-fun work might be necessary.

Recalling that almost all solutions are conceptually similar, it is assumed that a common model and associated APIs can be defined which are generic enough to cover the required functionality.

The mission statement therefore is:

- To define a common model for clustering on Linux.
- To define and implement a standard set of APIs for these on the Linux platform.

[7] As the focus is on HA clustering, MPI does not count.

3.2 Scope

While the primary focus is Linux, the standards aspire to be platform-agnostic; as stated above, our search for inpedendent standards on other platforms was negative, so they might benefit as well.

We try to keep the standard open for input from the High Performance Computing community too, but the bias is towards High Availability. Fortunately, the overlap in the requirements is large enough that we could accommodate both sides so far.

3.3 Requirements

3.3.1 Overall project

The project as a whole has the following requirements:

- Bandwagon effect; capture enough mindshare that a substantial piece of the market adopts the standard; otherwise the work is mood.
- The work must be royality and IP free. We do want to support both Open Source and proprietary implementations.
- Timely results. It was felt that early release of something usable and iteratively improving upon this was important in preserving interest and building momentum.

3.3.2 APIs

For the APIs[8] to be generally accepted, they must meet the following requirements:

- Wherever possible, the APIs shall adopt best practices of current implementations or be abstract enough to support as many of them as possible.
- The APIs, though primarily targeted at Linux, should not be applicable to Linux only.
  - Linux implementation is the primary target. Supporting another operating system should not negatively impact the Linux implementation.
  - Nevertheless we do want to define the APIs so as to make it as easy to implement in other operating systems as is consistent with the first point; it was mentioned that the APIs need to be OS-independent to be useful in HPC work, where applications are often written by people who don't own the clusters they will run on, and are often run on more than one cluster.
- API specs should aspire to POSIX compliance; in general, they should extend existing specifications with cluster functionality if sensible.
- APIs should in no way dictate an *in-kernel* or *user-space* implementation exclusively.
- Consistency.
- While every project is free to implement these APIs directly, it should also be possible to provide a compatibility layer on top of legacy code.

[8] In this document, the terms *API providers* and *consumers* are used respectively.

3.3.3 Reference implementation

The following requirements for the reference implementation have been put forward:

- The build system should be chosen to build on various platforms[9].
- Components should be portable wherever possible.
- Component interfaces should be agnostic with regard to OS and to kernel vs. non-kernel implementation.

3.4 History

While the problem has lurked in the mind of people for a long time already[10], the issue has not received enough attention until recently, when the pain became noticeably worse.

The success of the *heartbeat-stonith* library, now used by at least three clustering packages, also suggested that the time might be right now.

At the *Ottawa Linux Symposium 2001* an introductionary meeting took place, where many people got together and discussed the problem; as a follow-up, a three day workshop was held directly prior to the *Linux Kongress 2001* in Enschede, where the general direction was agreed upon and the first cut at the proposed component model was outlined – see [GL01].

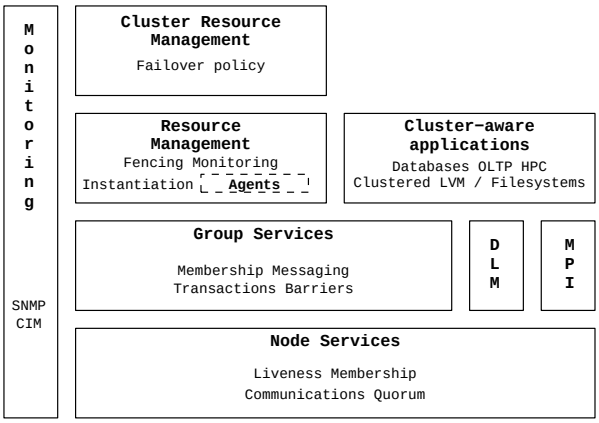
Since then, there has been a lively discussion on the mailing list, a website has been setup and more and more people, projects and companies are getting involved every week.

In 6, the author tries to give a prediction of where OCF is going.

4 The current component model

4.1 Overview

The current proposal for the functional components of the Open Clustering Framework is shown in the following diagram; it is based on our experience with how common clustering software is structured. It reflects the current status of our discussion and is subject to change.



The components are broken up according to the *objects* they deal with; *nodes*, *process groups*, *locks* et cetera. The APIs we are preparing will map to the external interfaces of these components.

A given implementation may chose to implement only one of these components or a subcomponent, using the other components at will.

However, the implementation might also provide the functionality of more than one component in one brick, and use whatever internal structure it desires. As long as it exposes these APIs, a cluster-aware application or an other

[9] i.e., autoconf

[10] See [HM97], section thirteen, or [GP97], section 15.3.

component will be able to use it nonetheless.

This allows the programmer to focus on his main area of interest and expertise. Within certain limits, an system architect will be able to build a system from the best-fitting components.

As components can be exchanged, the need to build the one-size-fits-all solution also diminishes; for example, a two-node cluster requires sufficiently less complex node services than a cluster of arbitrary size and may be configured more easily. The architect can chose the proper component for the intended use.

4.2 Node Services

Node services encompass all services relating to nodes.

One of the open questions here is how to identify a specific node; the main conceptual difference is whether nodes are identified by *host names, an integer sequence number* or even their primary *IP address*. All of these have precedents and good reasons in their favour, and specific optimizations the higher layers or applications can implement if they know about it.

This demonstrates one of the issues we are facing; namely, level of abstraction in the API and the model. We have to cope with all of the possibilities reasonably well, so the *node identifier* will be either treated as an opaque blob in the API and the API consumers will have to deal with it, or different kinds of clusters might require a recompile of the consumers.

4.2.1 Node Liveness

The component responsible for monitoring node health; mostly implemented as a binary *alive* or *dead*, it could also yield finer-grained data.

4.2.2 Node Communication Services

The lowest level of the cluster communication services; the instances of the cluster software on each node need to talk to those on the other nodes.

This is potentially used not only by the node membership services but also the higher-level group communication services.

The challenge will be to define a lowest-level denominator which can abstract the different models used for authentication, encryption, messaging and addressing semantics.

4.2.3 Node Membership Services

The membership – the list of currently active or eligible nodes – in a cluster is always in flux, because outages are only detected asynchronously. However, it is vital that the members have a coherent picture of the cluster; this component uses the *Node Liveness* data to compute the current consensus membership.

4.2.4 Node Quorum

In a high-availability cluster, one of the most important questions is who is allowed to modify – in any way – the shared resources and

the data. In the case of a so-called cluster partition, where part of the cluster loses contact to the rest, the cluster has to arbitrate between the fragments, ensuring that a maximum of one partition continues to operate on the data.

A quorum is *the number (as a majority) of officers or members of a body that when duly assembled is legally competent to transact business*, and this – majority of eligible cluster nodes survives – is exactly how this is often achieved, but implementations where the *tie-breaker algorithms* take additional inputs such as access to a specific device are also common and in fact required in case of evenly split clusters where a majority does not exist.

Most clusters implement the notion that only one group of nodes has *quorum*; this is a globally exclusive. However, this breaks down for large and potentially hierarchial clusters, where multiple levels of quorum exist; the discussion is not yet closed on this topic.

4.3 Group Services

A cluster-aware application knows that it is being run in a distributed environment, spanning multiple nodes. A common model for this is that the application forms a *process group* across the cluster nodes, hence the name of this component.

4.3.1 Group Membership Services

This is very similar to the node membership services, just a layer above – the group also needs to know the list of processes joined and be informed of leaves and died processes.

4.3.2 Group Quorum Services

Discussions have also begun on deciding whether quorum is only a property of the node layer, or whether different process groups can have different ideas of whether they have quorum or not; good points have been put forward for both cases. While current clusters mostly only provide node quorum, the framework should be flexible enough to be extended to allow for group quorum, too.

4.3.3 Group Messaging Services

The group messaging service provides communication services for the process group.

Facing the same challenges as the node communication services, this is also further complicated by the fact that group messaging often has additional guarantees with regard to ordering of messages – *virtual synchrony*, where all messages in a group are globally ordered. Not all messaging services have this property, though.

While the syntax can be the same in both cases, the semantics and guarantees are different; it has not been decided yet how to cope with this.

4.3.4 Group Voting Services

A group very often has the need to vote; for example, to agree on a primary coordinator for performing a particular operation.

4.3.5 Group Barrier Services

A distributed process group also needs to ensure that certain operations are only started if all members of the group have reached a specific state; this state is called the *barrier*.

The barrier services will provide an easy means to implement such a synchronization mechanism.

4.3.6 Group Transaction Services

Transactions are a common approach to providing fault resilience; a transaction either commits as an atomic, correct transformation of the system state or is rolled back completely, the system is always in a valid state.

For clusters, distributed transactions with two-phase commit are especially useful. A *transaction monitor* initiates the transaction and leads the clients through, ending with either a *commit* or *rollback*. By utilizing the transactional framework, the distributed application inherits transactional properties. Journaling, logging and recovery are simplified compared to home-grown solutions.

The transaction framework itself is very generic; only the *resource managers*[11] vary. At the same time, it is also very complex – it has to deal with a multitude of failure modes and guarantee the *ACID*[12] properties for all of them – and this implies that a well-tested component which can be shared between different implementations is very desirable.

For a more detailed treatment of transactions, see [Gray93].

[11] Not to be confused with the resource concept used in fail-over systems!

[12] Atomicity, Consistency, Isolation and Durability

4.4 Resource Management

High-availability clustering, especially the so-called fail-over or switch-over systems, mostly centers around managing groups of resources.

A resource is a single physical or virtucal entity that provides a service to clients or other resources. For example, a resource can be a single disk volume, a particular network address, or an application such as a web server. A resource is generally available for use over time on two or more nodes in a cluster, although it usually can be allocated to only one node at any given point in time.

These basic resources are combined into resource groups, which are treated as atomic units by the fail-over system; moving only the IP address but not the database itself would be devastating.

4.4.1 Resource Instantiation Facility

If the cluster decides to online or offline a resource on a specific node, a generic mechanism for doing so must exist; in the most simple case, this could be an abstraction to provide *Secure Shell*-like functionality in a portable fashion.

4.4.2 Resource Monitoring (RWATCH)

Resources also have to be monitored for health; fail-over solutions which only deal with health at the node level are less useful, as approximately eighty percent of all failures are due to software.

This sub-component should provides a generic interface for polling the health status of a resource or notifying the cluster that a resource has failed.

4.4.3 Resource Fencing Services

As explained above, most resources that are not designed to be cluster-aware only support being active once at a time; havoc and data corruption will result otherwise.

The fencing sub-component ensures this by allowing the cluster to enforce policy with regard to access to shared resources. Various levels of granularity exists; some clusters will radically fence a failed or partitioned node from all shared resources by flipping its power switch[13], while others might support fine-grained control – see [Brower00] for a proposal.

Work has already begun on this sub-component; the *heartbeat* package by Alan Robertson includes a *STONITH* library for driving various power switches, which is a very effective, albeit brutal, way of ensuring that a node stops accessing shared resources immediately.

This sub-component is also related to node or group quorum, as some systems treat quorum as just another resource which a cluster partition either has or does not; if one side fully fenced the other, it can be sure that only itself has quorum and can proceed.

[13] *Shot the other node in the head*, often abbreviated to *STONITH*

4.4.4 Resource Agents

Resource Agents are the glue layer between the switch-over software and the actual resources being managed. They aim to integrate the resource with the switch-over software without any modifications to the actual resource provider itself, by encapsulating it carefully and thus making it movable between real nodes in a cluster.

They are obviously very specific to the resource type they are encapsulating, however there is no reason why they should be specific to a particular switch-over solution.

All resources have a common set of methods which they must expose to the fail-over software; starting, stopping, a status query; this mostly maps one-to-one to Linux Standard Base init script functionality, with the exception that multiple instances of the same type can be active on a given node simultaneously, uniquely identified by the resource instance parameters and that they are usually much more paranoid than init scripts.

These *resource agents* are a common concept among all switch-over solutions; they directly map to the range of applications supported by them.

However, once more, the actual interfaces vary in details, which means that the ISVs cannot provide a common plug-in with their application for all switch-over clustering solutions as they should, because hopefully nobody knows better how to control their application than the ISV, but instead the vendors of the clustering solutions usually have to provide this code themselves.

The current, already rather complete, draft for the interface between the *Resource Agent* and

the switch-over software can be found on *on the OCF website*[14].

4.4.5 Cluster Resource Manager

If the node membership changes, a resource monitor reports failure or the administrator requests it, the *cluster resource manager* coordinates the recovery of the resource group; either locally or by transition to another node.

It is also responsible for ensuring exclusivity as explained under *Resource Fencing*.

4.5 Distributed Lock Manager

Many cluster-aware applications coordinate access to shared resources by locking the objects in question; this is a common approach to ensuring coherency and synchronization. Recovering locks in case of failures is a very complex topic and probably these are the hairiest programs in existence.

Due to historic reasons, almost every lock manager provides an interface both syntactical and semantically very similar to the VAX cluster lock manager, so we have a good precedent for the API here. It is assumed that just minor adjustments will be made to ensure a coherent design in the API.

One example of an Open Source distributed lock manager can be found *on IBM's website*[15]; it is already using *heartbeat* for the cluster infrastructure.

[14] http://www.opencf.org/standards/
[15] http://oss.software.ibm.com/dlm/

4.6 Cluster Monitoring

After a coherent API for a component has been defined, it will also be possible to have a common interface to *Network Management Systems* for all clusters on Linux.

An excellent case is the definition of a SNMP MIB for monitoring the cluster; basically functionality – exporting the current membership view, sending traps in case of membership changes et cetera map one-to-one to the APIs being discussed.

4.7 Cluster Configuration

Another complex issue is the configuration of a cluster; many approaches from a system administrators point of view have already been tried. This is not currently being discussed as part of the Open Clustering Framework for now.

4.8 The glue

4.8.1 PILS

Many modern Linux systems make extensive use of dynamically loadable object modules (plug-ins); this framework architecture is the perfect example.

However, most of these systems implement their plug-in and interface management in a way that satisfies their own immediate needs only, and is not generally directly usable by other projects.

PILS is an generalized and portable open source plug-in and interface loading system. PILS is being developed as part of the Open

Cluster Framework reference implementation; please visit Alan's talk for details on it.

4.8.2 Kernel to user-space and back

Component providers can live both in the kernel and in user-space; nevertheless, consumers from both environments need access to the functionality.

If a common API for both exists, and a component only offers one aspect, a generic layer should be able to translate between kernel and user-space calls and vice versa.

4.8.3 Generic event mechanism

Almost all of the components have the need to deliver events to and to react to events triggered by their consumers.

An initial, very well prepared draft by Ram Pai and Joe DiMartino was posted to the OCF mailing list in April this year and triggered a lot of discussion; it is available via the mailing list archives referenced from our website.

5 Affiliation with other groups

5.1 Free Standards Group

Our goal is to become a working group under the umbrella of the FSG; we have already begun work on this. Our initial draft for the answers to their *questionnaire for new working groups*[16] can be found *on the OCF website*[17].

We hope to benefit from their organizational experience and legal framework, and being able to concentrate on the technical work instead.

5.2 IEEE Task force on Cluster Computing

The IEEE does have a task force dedicated to cluster computing; however, their focus is primarily on High Performance Computing. The *subgroup working on High Availability*[18] *TFCC-HA* takes a passive stance and mostly monitors the development; they have shown interest in working together with the Open Clustering Framework group.

5.3 MPI Forum

The High Performance Computing community has their own well-accepted message passing interface standard. As it has different goals, it is not immediately adaptable to the needs of the High Availability community. However, the Open Clustering Framework should allow a MPI layer on top of it, so that HA and HPC applications can run in the same cluster.

5.4 Service-Availability Forum

The *SAForum*[19] is a group of companies aiming to provide *Open Standards for on-demand, uninterrupted communication services*, which is marketing-speak for saying that they are

[16] http://www.freestandards.org/policy/fsg102-newworkgroup-draft.txt

[17] http://www.opencf.org/OCF-fsg102-1.html

[18] http://www.csse.monash.edu.au/~rajkumar/tfcc/high-availability.html

[19] http://www.saforum.org/

aiming to provide much the same standards as OCF, just with an initial focus on the telco industry.

When OCF was created, we did not know about the SAForum; they appear to have formed roughly around the same time. However, we have tentatively begun to talk and found no major obstacles to a cooperation yet.

The main difference is that the SAForum is more closed, requiring a substantial entrance fee, even though the resulting standards are also supposed to be royality-free and should also allow an Open Source implementation.

The future relationship of the two efforts is unknown as of this time; the possibilities range from two totally disjunct efforts – which would be a waste, but even two standards is better than twenty-five – to a very close cooperation and potential joint working groups.

5.5 Members of the OCF group

It is kind of hard to answer this question at this point in time; as the Open Clustering Framework does not have a fixed organizational structure yet, no formal membership has to be requested or granted; as a consequence, while the list of members subscribed to our mailing list includes all the major players in the field, they cannot be listed here.

For an abbreviated list, please see *the OCF website*[20], or even better, visit the presentation itself, where you can meet the supporters in person.

5.5.1 The Linux HA Project

The *Linux HA project*[21] was founded by Alan Robertson; it provides *heartbeat*, the most well-known two-node switch-over solutions for Linux.

It is listed here in particular because its *heartbeat-stonith* library was the first component specifically targeted at clustering shared by multiple HA solutions – *heartbeat* itself, *Linux FailSafe* and *Kimberlite* – and because Alan has begun work on evolving *heartbeat* to a reference implementation of the OCF work.

6 Crystal ball gazing

The Open Clustering Group is beginning to capture vendor attention; as such, a more formal organization is likely required in the near future. We hope that this can be achieved in cooperation with the Free Standards Group.

Of course, at the same time, increasing mind- and market-share is very important; the cooperation with the FSG and the SAForum is an important issue here.

This is showing good progress; if a clustering vendors does not understand the benefits of standarization, their competition will gladly point them out to the customers. So everybody has an incentive to not be left out. As ISVs strongly benefit from this effort, their are putting their weight on it, too.

So far, we have not met someone who has not been supportive of the effort; everybody agrees it is required for Linux to succeed in the enterprise market, and that all parties involved ben-

[20] http://www.opencf.org/

[21] http://www.linux-ha.org

efit from it.

Work has begun on the standards; two drafts have already been produced. The next step is to provide Open Source implementations of these and convince vendors to also implement them. The first API which will see deployment is very likely the Resource Agent specification.

There is still a lot of work to be done both on the standards, the models, and on developing a common choice of words. Participation is actively invited[22].

Join and contribute now, while admission is still free!

7 Acknowledgments

Besides thanks to all participants of the Open Clustering Framework – you are too many to list one by one – the following documents were particularly helpful in the preparation of this paper:

References

[AlanR01] `Alan's paper` where he outlined the idea behind the Open Clustering Framework for the first time.

[HM97] *Linux High Availability HOWTO*[23] by Harald Milz. 10

[GP97] *In search of clusters* by Gregory Pfister. 10

[GL01] *Linux Kongress 2001 Workshop Summary*[24] by Greg Louis. 10

[Gray93] *Transaction Processing: Concepts and techniques* by Jim Gray, Andreas Reuter; published by Morgan Kaufmann. 12

[Brower00] *Resource fencing framework*[25] by David Brower; the initial draft posted to the *linux-ha-dev* mailing list in 2000; lots of discussions followed. 13

[22] As a blunt advertisement, we are in serious need of a webmaster for the OCF website.

[23] http://www.ibiblio.org/pub/Linux/ALPHA/linux-ha/High-Availability-HOWTO.html

[24] http://www.opencf.org/enschede2001/Enschede.summary.txt

[25] http://lists.community.tummy.com/pipermail/linux-ha-dev/2000-March/000394.html