Computer availability within a computer cluster

Björn Lindberg

Availability is when a resource at any given moment in time has the possibility to start a computational calculation without any delay.
Tillgänglighet inom ett datorkluster

Sammanfattning

Summary

This thesis handles the subject of availability within a Linux/UNIX computer cluster. What availability variables should be monitored and how to monitor them. By looking at what kind of systems most cluster administrators use and what kind of computational software they use within their cluster, we have made a list of important variables. The list of important availability variables that should be monitored are Domain Name System (DNS), Ping, Network File System (NFS), License and Remote SHell (RSH). If any of these variables does not work, the computational calculations can not start. These variables must be monitored in some way. We have used Nagios for this task. There is however two variables that Nagios can not monitor and these are license expiration date and RSH. The first installation of Nagios is time consuming, because it does not have any kind of (graphical) tool for editing the configuration files.
Innehållsförteckning

Sammanfattning ..................................................................................................................................................i
Summary .......................................................................................................................................................... ii
Innehållsförteckning ........................................................................................................................................ iii
Abstract ............................................................................................................................................................. 1
1. Introduction .................................................................................................................................................... 1
2. Background ................................................................................................................................................... 2
3. Computational environment ...................................................................................................................... 2
4. Selection of variables .................................................................................................................................. 3
   4.1. Variable summary .................................................................................................................................. 5
   4.2. Variable dependencies .......................................................................................................................... 5
5. Availability tool demands .......................................................................................................................... 5
6. Case study setup ......................................................................................................................................... 6
   6.1. Lab environment ................................................................................................................................... 6
6.2. Monitoring tool ....................................................................................................................................... 6
   6.3. Installation ............................................................................................................................................. 6
7. Result ............................................................................................................................................................ 6
8. Conclusion .................................................................................................................................................... 8
9. Miscellaneous ............................................................................................................................................... 8
10. Future work .................................................................................................................................................. 8
11. Acknowledgements ................................................................................................................................... 8
12. References ................................................................................................................................................ 8

Bilagor

A. Nagios screen dumps
Abstract

This thesis handles the subject of availability within a Linux/UNIX computer cluster. What availability variables should be monitored and how to monitor them. By looking at what kind of systems most cluster administrators use and what kind of computational software they use within their cluster, we have made a list of important variables. The list of variables that should be monitored are Domain Name System (DNS), Ping, Network File System (NFS), License and Remote SShell (RSH). If any of these variables does not work, the computational calculations can not start. These variables must be monitored in some way. We have used Nagios for this task. There is however two variables that Nagios can not monitor and these are license expiration date and RSH. The first installation of Nagios is time consuming, because it does not have any kind of (graphical) tool for editing the configuration files.

Keywords:
Availability, cluster, COTS, Beowulf, Nagios.

1. Introduction

When air moves around a formula 1 car it is a complex environment, the air moves with different speeds and pressures over different parts of the car body. In the old days the formula 1-teams calculated air flow and pressures around the car using pen and paper. This process was very time consuming, resource demanding and was prone to human errors. Then when the part was manufactured they tested it in a wind tunnel or put it directly on the car for testing on the track. If the part was not working correctly they created a new one and tested again. This trial and error was time consuming and costly. All this work was done to evolve the cars. Today all parts are tested in computer simulations before they even consider manufacturing the part. These simulations demand a lot of computing power. To get the necessary power they use computer clusters. A computer cluster is a number of computers (cluster nodes) that are connected through a computer network. The computer cluster gives the computational engineer a possibility to execute a calculation in parallel over a large number of computers simultaneously. By running these calculations on a cluster the time it takes to conduct the calculation is significantly reduced and the amount of human errors is lowered.

There exist many different software simulation packages for computational calculations. All these packages are specialised on a certain type of calculations, for example: Fluent [1] and CFX [2] are used for CFD-calculations, MSC.Marc [3] uses the FE-analysis method, MSC.Nastran [4] is used to among other things calculate structural loads and Ansys [5] uses a mix of methods to measure life length of an object.

A computer cluster environment within a company could be built up with a wide range of hardware and different operating systems. The hardware used in a computer cluster could be anything from super computers, made specifically for one specific task, to Commercial Off-The-Shelf computers (COTS), which can be used for different tasks. COTS clusters could also be called Beowulf [6] clusters. To connect computers in a cluster to each other, a wide variety of network products and topologies could be used. For example; InfiniBand, Myrinet and Ethernet, for the complete list of interconnect types read reference [7]. The most common operating systems used for computation calculations are UNIX [8] or Linux [8] based, but there are also some applications that work on windows based operating systems. A computer cluster needs an array of different support-servers to be able to administrate the computer cluster in an effective way. Some examples of support tasks are license-services, DNS-services [9] and NIS-services [10]. To acquire data, store data and access computational applications, many COTS clusters uses Network File System (NFS) [10] to transfer and share data between the cluster nodes.

All computational cluster environments need some kind of system for controlling availability. An availability system will help the computational engineers to detect if there are some problems with the cluster nodes they intend to use before they conduct the calculation. An availability system should also be able to warn cluster administrators about arising problem. Systems like this should also
be able to save data over time so it could create availability statistics based on the saved data.

The first goal with this thesis is to gather information about what variables are necessary for a computational calculation to start. For example are all the necessary service-supports up and running? The second goal for this thesis is to find some kind of tool or tools that can gather and present this information for the computational engineers and cluster administrators. The case study will be the base for recommendations of what tool or tools in combination that can gather and handle the important variables in a good way. The tool or tools chosen will be tested in a case study but it will not be implemented in real life within the boundary of this paper. The variables gathered in this paper will be a part of a future Service Level Agreement (SLA) even if the construction of the SLA will not be a part of this thesis.

2. Background

When the computer community writes about computer availability the answers is significantly different depending on who has written the text.

What is availability then? Tanenbaum & Steen states in their book [11] that “Availability is defined as the property that a system is ready to be used immediately.”. They mean that availability is when a system that can be used at the exact moment, when for example a user or a service needs the resource. William [12] states that availability is equal to “A measure of user’s ability to gain access to a computer system”, this is similar to Tandevenbaum & Steen’s definition. In the student material from HP [13] they defined availability as “Availability (%) – this is the ability of an IT service or infrastructure component to perform its required function at a stated moment or over stated elapsed period of time.”. So every given moment when someone or something tries to use a resource within the cluster and it is either working or not working it is called availability.

When some cluster administrators and users talk about their computer-cluster they speak about scalability, to increase the throughput under an increased load, but according to Buyya [14] so are “Availability is, in several cases, more important than scalability”. If the cluster has a low availability it might not be necessary to increase the numbers of nodes to increase the throughput. When we summarise the definitions and statements above, we can conclude that availability could be described as follows:

**Availability is when a resource at any given moment in time has the possibility to start a computational calculation without any delay.**

Availability could be measured in many different ways depending on what kind of demands a specific organisation has for the computer cluster. A simple way to control if the cluster nodes are working could be to ping them and see if they respond, but just pinging a cluster node does not show if it really is working correctly. Ping only shows that the network is up and running and that the network card and IP-stack probably working correct. Ping do not show if any process on the cluster node do not work as intended or if the cluster node for example is connect to all the necessary support-servers that are needed for the computational calculation to begin.

To measure availability some kind of tool or combination of tools are necessary. This tool or tools should be able to control all the necessary variables needed for every specific type of computational calculation to start, for example a MSC.Marc calculation requires response from the MSC license-server while a Fluent.INC calculation need an answer from the Fluent license-server.

3. Computational environment

A computer cluster is a complex environment with a lot of different hardware and software. All this different hardware and software needs to be able to work together like one single environment, without any flaws that could affect the main purpose of the cluster in a negative way. The aim for a computer cluster is to reduce the calculation time for the computational engineers without affecting the integrity of the calculation. A computer cluster does by default work in all OSI-layers [9] from layer one where the physical cables and currents flying through them, to layer seven where the computational calculation software are located.

What kind of hardware, infrastructure and software do we need to build a cluster that should be able to run a variety of different computational calculation software? To answer this we need to know what kind of computational calculation software we will use and what demands they have on the hardware? When we have figured out what software to use and its demands we can decide what kind of cluster we want to build. In this example we will build a Linux based COTS cluster.

So what is needed to build a basic COTS cluster? We need a room with enough electrical power and cooling capacity to be able to power and cool lets say 200 3.6 GHz, 64-bit Intel based computers. The computers must have some kind of connectivity and in this case we use 1Gbps ethernet [9], TCP/IP [9] based network, built with switches in a star topology [15]. We also need some kind of support-servers that is necessary in a cluster like this. In figure 1 we have drawn a schematic figure of this

---

2006-06-07, 2(10)
cluster. All the cluster nodes and support-servers are confined within the same graphical location.

![Figure 1 schematic COTS cluster](image)

Now when we know what kind of computers, network and computational calculation software we will use, we have to figure out some other basic features that are needed in this computer cluster.

The computers need some kind of name and IP-addresses. We will give them a static IP-address. The cluster nodes names will be distributed with the help of a DNS-server.

We need some kind of authentication system to reduce the amount of work it takes to administrate user rights. In this case we will use NIS as the authentication system. NIS demands that all the cluster nodes and servers are running the service YPBIND and that the server is running the NIS daemon.

The computational software does in most cases demand some kind of license and one of the ways to utilize licenses in a good way is to use some kind of server that has a license pool of licenses that is handed out when a user need the license for a specific calculation. In this case we use FlexLM [16].

We also need some kind of utility to transfer and share data between the cluster nodes and servers, in this case we will use NFS. We will have specific servers for all the users home directories, computational calculation applications and project shares where the input and output from the calculations are stored. NFS demand that all the cluster nodes and servers are running the services PORTMAP and NFS.

Some computational calculation software also needs Remote SHell (RSH) [17] when they perform their calculations.

To summarize we need basic hardware like cluster nodes, servers, switches and cabling we also need power, computational calculation software, NIS-, NFS-, license -servers, and RSH.

### 4. Selection of variables

In section 3.0 we have described the necessary services needed to start a computational calculation. In this section we will look into these individual variables more extensively. We will describe if it is possible to get viable availability numbers from these variables.

**Power:** To be able to start the cluster computers and necessary network equipment, power is paramount, without power nothing will work. How to measure if all the cluster nodes and support-servers have power could be very difficult. One way could be to connect some kind of tool to the electricity-central and by that measure if the power is working. If the monitoring computer or the necessary networking equipment does not have power we can not gather any availability data anyway. Another way to measure power could be to ping all the computers, if we get a response from the ping the computer is up and running, but measuring power this way has the exact same problems as the previous way to measure power.

**Network:** When looking into network availability we have decided to split it into two parts. Part one; availability from the computational engineers perspective and part two; availability from the cluster maintenance personnel perspective.

From the computational engineers view good availability is when they have connectivity to all cluster nodes within the cluster and when they can start the computational calculations without any problems. Bad availability could for example be when one switch is down. In figure 2 they can just reach 50% of the cluster nodes if switch 1 is down and therefore they have just 50% access availability to start computational calculations. If switch 2 goes down they have 0% availability. Some cluster nodes could also be down but since the switch already is down there is of no consequences for the computational engineers.

From the cluster maintenance personnel perspective the availability issue is more complex. For example if a computational engineer calls the maintenance personnel and state that he/she can not start any new calculations. Then if only the network is down the maintenance personal only has to handle this problem. If the nodes also are down, the situations become more troublesome. In this perspective the cluster nodes could have 0-100% availability while the network has 0% availability.
OSI-layers are working correctly. From OSI-layer four it does not control if all seven software or hardware problems. RTT only works on the switches or there might be more severe problems with ping are to control if there is high load on the network by monitoring Round Trip Time (RTT). RTT controls the time it takes for an ICMP [9] packet to travel to and from a computer. If the RTT value is high there might be high load on the switches or there might be more severe software or hardware problems. RTT only works from OSI-layer four it does not control if all seven OSI-layers are working correctly.

RSH: Some of the calculation software needs RSH for communication between the cluster nodes and servers. Therefore it is important that RSH work in a correct manner. RSH is also a good way to control at least three dependencies as we describe later on in figure 3. The chain of dependencies shows that Remote Shell (RSH) [17] is high up in the chain. 1) Connectivity, 2) NIS-service and 3) DNS-service must be working. This is because RSH needs DNS to resolve the hostname and the NIS-service to confirm that it has the correct rights to connect to that cluster node or server and by default it needs connectivity. When we control that RSH works it also control all seven OSI-layers at the same time.

NIS: In a computer cluster we need some way to authenticate users, computers and services. Authentication can be done in many different ways for example using the server/client based Network Information Service (NIS) or Lightweight Directory Access Protocol (LDAP), these protocols are used to reduce the administration, if the services NIS and LDAP does not work as intended the computational calculation can not start. We can also use a system with, local user and password text files but as the cluster grow in size this will be hard to administrate.

There are several ways to control NIS availability; the first way is to control that the NIS daemon on the NIS-server is running (RPC.NISD). Remote Procedure Call (RPC) [9] is built in to certain programs, it basically responds to queries. The second way to do the controls is either from all the cluster nodes toward the NIS-server or from the monitor server towards the NIS-server, if the NIS-servers respond it is up and running. Controlling the name resolution from every cluster node would be optimal but it would also create a lot of network traffic, so the most reasonable way to control NIS availability would be from the monitoring server.

NFS: Network File System (NFS) is used in many UNIX/Linux networks to transfer and share data between the cluster nodes and servers. NFS needs every cluster node and server to have two daemons up and running (NFS, PORTMAP). It also requires that all AUTOMAP points are set up in a correct fashion and that MOUNT is able to use auto mount towards the correct mounting points. NFS also demands that NIS is up and running correctly so it can authenticate it towards the specific shares, if the services NFS and PORTMAP does not work as intended the computational calculation can not start.

We could control availability in the same way with NFS as with NIS above. To logon to every cluster node and try out all auto mount shares, this would create a lot of overhead and be practically impossible. Another way would be to within a time interval from the monitoring server do these auto mount checks. If the auto mounts checks work from the server we can conclude that probably all nodes also use auto mount.

DNS: The Domain Name System (DNS) is used in many network environments using TCP/IP to resolve domain names in plain text to IP-addresses. If the DNS-service is down it will cause trouble for the computer nodes to resolve the necessary IP-addresses needed to communicate.

License servers / service: Some of the computational calculation software demands some kind of licenses that the manufacturer supplies. For easy supervision and good utilization of the licenses some kind of license-service is needed. FlexLM is often used as the license-service which handles all the different license-pools. When a computational engineer starts a calculation, the software needs a license from the license-service. The licensing-service is in many cases load balanced over more then one server. These servers load balance the licenses so if one server goes down there is always at least one server who runs the license-service and hand out the licenses.
What license variables should be considered when monitoring availability then? 1) Is there enough license-tokens left to start the calculation? 2) That the licensing-service is up and running. 3) That the final expiration date of the particular license does not run out during the calculation duration period. We consider that the tokens left issue belongs to capacity handling and that the issues with final expiration date and the fact that the licensing-service is up and running should be counted as availability issues.

Measuring license availability is tricky because the service is load balanced between one or more servers. Should we measure the license-servers one by one and say that if all license-servers are working at 100% and that the license-services is working in a correct manner we have 100% availability? We think this way is wrong because if one of the servers is down the license-service still have 100% availability. So availability on a load-balancing license-service should be measured in a combined number. For example if we have three license-servers all running the same license-service and one of the licensing-servers goes down the availability should still be 100%.

4.1. Variable summary

In table 1 we have collected the important variables possible to monitor and that would give a necessary availability numbers.

Table 1 variables to control

<table>
<thead>
<tr>
<th>Variable</th>
<th>Control (from server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS</td>
<td>Resolve a common name with a IP-address</td>
</tr>
<tr>
<td>PING</td>
<td>PING to all cluster nodes and servers.</td>
</tr>
<tr>
<td>RSH</td>
<td>RSH to all cluster nodes and servers.</td>
</tr>
<tr>
<td>NIS</td>
<td>RPC control that RPC.NISD is running on NIS servers.</td>
</tr>
<tr>
<td>License (port up)</td>
<td>Control if the specific TCP port connected to that license is up and running.</td>
</tr>
<tr>
<td>License (expiration date)</td>
<td>Run a script that connects to license servers and grep necessary data date from the license.dat files.</td>
</tr>
<tr>
<td>NFS</td>
<td>RPC control that NFS is running on all cluster nodes and necessary servers.</td>
</tr>
<tr>
<td>NFS</td>
<td>RPC control that PORTMAP is running on all cluster nodes and support-servers.</td>
</tr>
</tbody>
</table>

4.2. Variable dependencies

As we have written in section 3 a computer cluster is a fairly complex environment where many services have to work together in a chain. The calculation engineer will not be able to start any calculation if any link in the chain stops to work. In section 4 we have dissected the variables, in section 4.1 we have collected the important variables into table 1. In figure 3 we have written down the chain of events that must work correct for a user to be able to start a calculation. We have also divided the chain into two areas, connectivity and application.

5. Availability tool demands

This section contains specific demands on tools used within a COTS cluster environment. Monitoring availability in a computer cluster must not in any way influence the computational calculations in negative way. In a paper written by Marzolla and Melloni [18] they have stated six different criteria for a monitoring tool. The first criteria are; intrusion-free the tool must not under any circumstances affect the result from the calculations. The second criteria are; low overhead the tool can not use a lot of computer recourses like CPU and network capacity. The third criteria are; batch operation the tool should be able to run in the background without any hands on work by the user or administrator. The fourth criteria are; generality the tool should be able to handle a lot of different network devices. The fifth criteria are; easy configuration the tool should be easy to maintain and configure. The sixth criteria are; reasonable scalability the tool should be able to scale good without overhead or workload for computers and administrative maintains should not grow. These six criterias are very important and we also have a few more criterias the tool or tools should be able to fulfil for example; save data in a database so we later on can use the data as proof in a SLA. Display data in some kind of GUI preferably web based. In the paper [19] by Matthew, Massie, Chun and Culler they have listed similar key design challenges as in the BaBar paper. The key challenges are scalability, robustness, extensibility, manageability, portability and overhead.

So what demands should we have for a tool or tools matching our schematic cluster in figure 1?
• The tool should not in any circumstances affect the performance of the cluster-computers and the network.
• The tool should not in any circumstances affect the calculation result in any way.
• The tool should be able to handle a different variety of network equipment and computer-protocols and operating systems.
• It should scale reasonably well within the confinement of the local area network (LAN) [9].
• It should run in the background and be able to run custom scripts.
• It should have some kind of graphical GUI and administration interface.
A tool controlling availability must handle all the criterias listed above.

6. Case study setup

In this section we describe our physical lab environment. The tool we choose to test and why we picked that tool. We will also make an extensive test on the chosen tool.

6.1. Lab environment

Our lab environment is built up by 5 Dell Optiplex GX620, (3.2GHz EMT64 CPU, 1GB RAM-memory) connected to a Cisco 4948 1Gbps switch. The operating systems on the two servers are Red Hat RHEL3-ES (kernel 2.4.21-37.el) and on the three cluster nodes we used Red Hat RHEL3-WS (kernel 2.4.21-32.el). T001 and t002 are the servers for the availability monitoring tools, while t003-t005 will act like cluster nodes in the availability system (see figure 4).

6.2. Monitoring tool

There is an abundance of (free) monitoring tools all with a mixed amount of functions and features. Some cluster owners have developed their own monitoring/availability tools like Marzolla and Melloni have done with their BaBar INFN Computer Cluster [18]. Other cluster owners use a mix of (free) Linux based tools to get the features they want.

What kind of features do we want then? We need a tool that can monitor the availability variables stated in table 1 and handle the demands stated in section 4. It should also be able to display the gathered data in a comprehensive way for both users and administrators. Another feature the tool should have is that should be able to save data for a longer period of time so it could be printed out and used in connection to a SLA.

We have chosen Nagios [20] as the monitoring tool because it can handle most of our demands. The reasons for choosing Nagios has been discussed in further detail in [28].

6.3. Installation

Nagios v2.2 [20] has some pre install requirements. We used Apache v2.2.0 [21] as web server, and we also installed GD-library v2.0.33 [22] and Perl v5.8.0 [23], [24].

When all the pre install requirements were fulfilled we installed Nagios and the official Nagios plug-ins (nagios-plugins-1.4.3.tar.gz). Guiding us during the installation was the recommendable book [25] written by Turbull. This book is comprehensive; it covers many areas of Nagios and has good examples and explanations.

When Nagios was installed we turned our attention on how to monitor hosts, but the default version of Nagios does not have any GUI interface to edit the configuration files with, thus we where forces to edit them by hand. We soon discovered that the configuration files where quite complex. We could for example create host and service templates, and then when we created a host we used the host template, and the result is that the host inherited the settings specified in the template. All the different ways these configuration files inherit each other is complex, so studying how they work is important before starting to configure anything. At the Nagios web page there is a good map over all the configuration files dependencies [26].

7. Result

We configured some local control checks with Nagios on our availability monitoring server. We checked ping towards local host, current load, current users, current swap size and current root partition size. Adding hosts and configuration services and bind these host and services together was a simple task.

Most built in check commands also have built in warning and critical checks. Different check commands have different warning and critical
values. If we for example control the size of our local root partition we could set warning at 20% and critical warning at 10%, Nagios will send out a mail to the cluster administrator informing him/her that the hard drive is almost full.

After the first local checks we looked into ways to check the variables listed in table 1.

**DNS:** The `check_dns` command control if resolving common domain name in plain text to an IP-address works. This check worked great we typed in for example `001` and it resolved IP-address `192.168.1.11`. The DNS check should be checked from the monitoring server.

**PING:** The `check_ping` command controls if the monitoring server gets response from all network equipment selected. The ping check should be checked from the monitoring server.

**RSH:** There is no built in check for RSH but it would not be any problem to build a check command in Perl. A check RSH command should basically connect to a host and run the command `hostname -r`, and then see if it responds with the same name we used to connect with, could probably add warning and critical check on response time. This check should be checked from the monitoring server.

**NIS:** There is a built in command called `check_rpc` this check command can control any kind of RPC applications like the RPC.NISD server daemon and YPBIND. The NIS check should be checked from the monitoring server towards the authentication-server, and if every cluster node is running YPBIND it should also be checked in the same way as the server daemon.

**LICENSE:** There is existing check command built into Nagios that can check any TCP port. But this command can only solve half of the problem and that is checking if the TCP port is up and running, and by that knowing that the license service is up. The problem is gathering the licenses expiration date. In this case study we have FlexLM as license service. This service is installed on several licensing-servers and many of these servers uses load balancing.

To get the data necessary we could have built the necessary (Perl) script. The script should connect to the licensing-servers and basically use the `grep` [27] command to gather necessary data from the license.dat files, and then display them in some nice fashion. We have not looked into this, because the licenses system is very complex with one main license and lots of module licenses in different configurations. This script should be run from the monitoring server connecting towards the licensing-servers.

**NFS:** This is also an RPC service just like NIS so we could easily gather necessary data from NFS and PORTMAP. The NFS check should be checked from the monitoring server towards all the NFS-server.

We have decided to gather all availability data from the monitoring-server connecting to necessary networking equipment. If all cluster nodes in the cluster should do the same availability check as the monitoring-server the network would grind to a halt. How often should the variables in table 2 be pulled then? Most of the monitoring tools we have read about have five minutes as a default value and among the few papers we have found where availability is mentioned no other value has been expressed. The license expiration date does only need to be controlled once a day, simply because we only measure the date where the license runs out and that will not have constant changes. Nagios polls the data every five minute as default. We have not seen any reason to change this number when this case study was executed, but in the future this number should be looked at because checking all variables as often as five minutes is probably not necessary. In paper [28] by Lyshaugen this is looked at more extensively.

### Table 2 Nagios check commands

<table>
<thead>
<tr>
<th>Variable</th>
<th>Control command</th>
<th>Checks</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS</td>
<td><code>check_dns</code></td>
<td>if server is up or down. Warning and critical warning can be set up based on response time.</td>
</tr>
<tr>
<td>PING</td>
<td><code>check_ping</code></td>
<td>if host is up or down. Warning and critical warning can be set up based on response time.</td>
</tr>
<tr>
<td>RSH</td>
<td><code>none</code></td>
<td></td>
</tr>
<tr>
<td>NIS (YPBIND)</td>
<td><code>check_rpc</code></td>
<td>if RPC daemon YPBIND is up and running.</td>
</tr>
<tr>
<td>NIS (RPC.NISD)</td>
<td><code>check_rpc</code></td>
<td>if RPC daemon RPC.NISD is up and running.</td>
</tr>
<tr>
<td>License (license port up)</td>
<td><code>check_tcp</code></td>
<td>if service is up or down. Warning and critical warning can be set up based on response time.</td>
</tr>
<tr>
<td>License (expiration date)</td>
<td><code>none</code></td>
<td></td>
</tr>
<tr>
<td>NFS (NFS)</td>
<td><code>check_rpc</code></td>
<td>if service is up or down. Warning and critical warning can be set up based on response time.</td>
</tr>
<tr>
<td>NFS (PORTMAP)</td>
<td><code>check_rpc</code></td>
<td>if service is up or down. Warning and critical warning can be set up based on response time.</td>
</tr>
</tbody>
</table>
8. Conclusion

In this paper we have described the most common systems used in a Linux/UNIX cluster based on an array of papers, Beowulf archive [29], [30] and experience of our case study cluster. The study and experience from the case study have helped us fulfil the goals with this paper.

The first goal was to gather information about which variables that are necessary for a computational calculation to start. Finding these variables has been an interesting task, because no COTS cluster is built up in the same way. We have compiled a list of general variables (services) that should be monitored for their availability within a computer cluster. These variables have been divided into two areas, connectivity and application.

When controlling availability connectivity, ping is a good tool. Ping does in some cluster implementations need the DNS-service for resolving plain text domain to a valid IP-Address.

In the application part of availability we have placed three variables (services). Variable one is some kind of authentication service. One of the most used authentication systems is NIS. Variable two is the licensing system where FlexLM is one of the most used licensing software that is used within UNIX/Linux computer clusters. Variable three is that all cluster nodes and servers within a cluster need RSH. Basically the computational engineer can not start the computational calculation, if power, DNS, Ping, License, NFS or RSH fails.

For more information about these availability variables, that should be monitored, read sections 4.0 and 4.1. In figure 3, we have visualised how the different variables depend on each other.

The second goal with this paper was to find some kind of tool or tools that can monitor these availability variables. The tool must be able to monitor all variables in table 1 and save this data for later use in a SLA. We used the tool Nagios that can monitor almost all variables, the default Nagios can not monitor the license expiration date or detect if RSH is working correct. These flaws with the tool could however be fixed by writing our own scripts that handles these functions, and then display the data with help of Nagios. These scripts are not a part of this paper, but it would not be hard to develop.

There are also a couple of other negative things with the default version of Nagios. One of them is that it is time consuming to install Nagios the first time. This because it does not have any feature for editing all the configurations files with help of a web based GUI. Another issue is that it does not have any feature for creating network load graphs. Both these issues could be solved by adding extra plug-ins [31].

9. Miscellaneous

This paper is best read and understood if you have access to the other two parts of this three way split project. The three projects were conducted by three different persons handling each of their parts. Part two, by Thomas Lyshaugen, is about Capacity handling [28] and the last part, by Morten Dalhaug, is about Resource Handling [32].

10. Future work

For the future it would be desirable to develop the RSH and licence check modules in Perl. This is necessary to check all the variables stated in table 1.

Another issue is to check if it is possible to create dependencies in Nagios. These dependencies are necessary when displaying license-service availability, this because the license-services are load balancing. All three license-services serving the same license should have a combined availability number. For example if one of three license-services for Fluent stops working the calculation software still have a 100% availability from the other two, but what happens if a second Fluent license service also fails? This should be looked into more extensively.

Look further into how often data should be polled. Is five minutes to often or too seldom?

11. Acknowledgements

We want to send our thanks to our supervisor Andreas Boklund and our examiner Stefan Christiernin at the Division of Computer science at University West, Trollhättan, Sweden.

We also want to thank Susanne A., Göran T. and Christian S. for all the help and the opportunity to make our exam thesis, thanks.

12. References


2006-06-07, 8(10)


### Service Status Details For Host '001-localhost'

<table>
<thead>
<tr>
<th>Host</th>
<th>Service</th>
<th>Last Check</th>
<th>Duration</th>
<th>Attempt</th>
<th>Status Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Status</td>
<td>2006-05-16 11:38:57</td>
<td>156.01 s</td>
<td>5</td>
<td>DNS: OK, 0.012 s response time, 1.001 s return</td>
</tr>
<tr>
<td>001</td>
<td>License Check</td>
<td>2006-05-16 11:37:34</td>
<td>156.01 s</td>
<td>5</td>
<td>TCP: OK, 0.012 s response time, 1.001 s return</td>
</tr>
<tr>
<td>001</td>
<td>License Check</td>
<td>2006-05-16 11:37:39</td>
<td>156.01 s</td>
<td>5</td>
<td>TCP: OK, 0.012 s response time, 1.001 s return</td>
</tr>
<tr>
<td>001</td>
<td>License Check</td>
<td>2006-05-16 11:37:45</td>
<td>156.01 s</td>
<td>5</td>
<td>TCP: OK, 0.012 s response time, 1.001 s return</td>
</tr>
<tr>
<td>001</td>
<td>License Check</td>
<td>2006-05-16 11:37:50</td>
<td>156.01 s</td>
<td>5</td>
<td>TCP: OK, 0.012 s response time, 1.001 s return</td>
</tr>
<tr>
<td>001</td>
<td>License Check</td>
<td>2006-05-16 11:37:50</td>
<td>156.01 s</td>
<td>5</td>
<td>TCP: OK, 0.012 s response time, 1.001 s return</td>
</tr>
<tr>
<td>001</td>
<td>License Check</td>
<td>2006-05-16 11:37:50</td>
<td>156.01 s</td>
<td>5</td>
<td>TCP: OK, 0.012 s response time, 1.001 s return</td>
</tr>
<tr>
<td>001</td>
<td>License Check</td>
<td>2006-05-16 11:37:50</td>
<td>156.01 s</td>
<td>5</td>
<td>TCP: OK, 0.012 s response time, 1.001 s return</td>
</tr>
<tr>
<td>001</td>
<td>License Check</td>
<td>2006-05-16 11:37:50</td>
<td>156.01 s</td>
<td>5</td>
<td>TCP: OK, 0.012 s response time, 1.001 s return</td>
</tr>
<tr>
<td>001</td>
<td>License Check</td>
<td>2006-05-16 11:37:50</td>
<td>156.01 s</td>
<td>5</td>
<td>TCP: OK, 0.012 s response time, 1.001 s return</td>
</tr>
<tr>
<td>001</td>
<td>License Check</td>
<td>2006-05-16 11:37:50</td>
<td>156.01 s</td>
<td>5</td>
<td>TCP: OK, 0.012 s response time, 1.001 s return</td>
</tr>
</tbody>
</table>

**Screen shot the availability monitoring server t001 and all its local control checks.**
### Computer availability within a computer cluster

**Appendix A**

#### Screen shot of the availability display breakdown of server t001

<table>
<thead>
<tr>
<th>Service</th>
<th>Time OK</th>
<th>Time Warning</th>
<th>Time Unknown</th>
<th>Time Critical</th>
<th>Time Undetermined</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS Check_http</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>License Check_Fern_CISCO</td>
<td>95.24%</td>
<td>95.24%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.75%</td>
</tr>
<tr>
<td>License Check_Fern_VMR</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>License Check_Fern_par_SLEEP</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>License Check_Fern_par_CRC</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>License Check_Fern_par_HMAC</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Local Check_Curr_load</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Local Check_Curr_Uptime</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Local Check_Disk_Partitions</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Local Check_Disk_Space</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Local Check_Total_Processes</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Local Check_Used_Ram_Memory</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>NFS Check_FSID_local-Service</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>NFS Check_FTIPM_local-Service</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>NFS Check_HTTP</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>NFS Check_ssh/Id2</td>
<td>58.82%</td>
<td>92.82%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.56%</td>
</tr>
<tr>
<td>NFS Check_TCP/Local-Service</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>NFS Check_NUM气血</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Average</td>
<td>95.92%</td>
<td>95.92%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

**Host Log Entries:**

***View full log entries***

<table>
<thead>
<tr>
<th>Event Start Time</th>
<th>Event End Time</th>
<th>Event Duration</th>
<th>Event Severity</th>
<th>Event Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006-05-16 01:00:00</td>
<td>2006-05-16 01:00:30</td>
<td>03:00:00</td>
<td>01:00:00</td>
<td>HOST UF VMSHE</td>
</tr>
<tr>
<td>2006-05-16 01:00:00</td>
<td>2006-05-16 01:00:30</td>
<td>03:00:00</td>
<td>01:00:00</td>
<td>HOST UF VMSHE</td>
</tr>
<tr>
<td>2006-05-16 01:00:00</td>
<td>2006-05-16 01:00:30</td>
<td>03:00:00</td>
<td>01:00:00</td>
<td>HOST UF VMSHE</td>
</tr>
<tr>
<td>2006-05-16 01:00:00</td>
<td>2006-05-16 01:00:30</td>
<td>03:00:00</td>
<td>01:00:00</td>
<td>HOST UF VMSHE</td>
</tr>
<tr>
<td>2006-05-16 01:00:00</td>
<td>2006-05-16 01:00:30</td>
<td>03:00:00</td>
<td>01:00:00</td>
<td>HOST UF VMSHE</td>
</tr>
<tr>
<td>2006-05-16 01:00:00</td>
<td>2006-05-16 01:00:30</td>
<td>03:00:00</td>
<td>01:00:00</td>
<td>HOST UF VMSHE</td>
</tr>
<tr>
<td>2006-05-16 01:00:00</td>
<td>2006-05-16 01:00:30</td>
<td>03:00:00</td>
<td>01:00:00</td>
<td>HOST UF VMSHE</td>
</tr>
<tr>
<td>2006-05-16 01:00:00</td>
<td>2006-05-16 01:00:30</td>
<td>03:00:00</td>
<td>01:00:00</td>
<td>HOST UF VMSHE</td>
</tr>
<tr>
<td>2006-05-16 01:00:00</td>
<td>2006-05-16 01:00:30</td>
<td>03:00:00</td>
<td>01:00:00</td>
<td>HOST UF VMSHE</td>
</tr>
<tr>
<td>2006-05-16 01:00:00</td>
<td>2006-05-16 01:00:30</td>
<td>03:00:00</td>
<td>01:00:00</td>
<td>HOST UF VMSHE</td>
</tr>
</tbody>
</table>

---

**Appendix A**

2006-06-07, 2(2)