the Availability Digest

www.availabilitydigest.com

Data Center Monitoring with Open-Source Nagios

Wolfgang Breidbach
Director, NonStop
BV Zahlungssysteme
November 2011

A primary requirement to achieve high availability is to be able to act proactively, not reactively. Problems that may be critical should be detected at the earliest possible time so that actions can be taken to correct them automatically, if possible, or to alert the operations staff with alert messages describing the problem so that manual action can be taken.

BV Zahlungssysteme, or BV Payment Systems in English, a subsidiary of Bank-Verlag GmbH, provides secure systems for card-based payment transactions and electronic banking for the member institutions of the Federal Association of German Banks. Our credit-card, debit-card, and online banking services must be always available, as their failure can bring retail commerce to a halt in Germany.[1]

The System Monitoring Challenge

To provide continuous availability, we at BV Zahlungssysteme operate two geographically dispersed data centers. The heart of our financial-service processing architecture is two HP NonStop servers in an active/active configuration.[2] Both systems are actively processing transactions. Should one fail, it is simply removed from the processing pool until it is restored to service. In 2012, this configuration will be expanded to four production systems due to a joint effort with the Cooperative Banks of Germany. In addition, we operate a QA system and a development system for NonStop support.

Supporting the NonStop servers are many Unix, Linux, and Windows servers. To keep this complex operational, it is imperative to be able to monitor all of the servers and the other data center components with a single system monitor – a "single pane of glass." There are many good monitors available for tracking the status of servers, storage subsystems, and networks, but these monitors in general do not support HP NonStop servers.

In today's world, a NonStop system can no longer be seen as a standalone system with its own management, operation and monitoring processes. Most datacenters recommend a central point of monitoring and/or service integrated into the first level support team.

Recently, we tried to monitor our many Unix, Linux, and Windows servers and, of course, our four NonStop systems with a vendor tool; but that solution did not meet all of our requirements. Importantly, the system management team responsible for our Unix and Linux servers was not satisfied.

[1] This article was originally published in The Connection in the November/December issue and is republished here with The Connection's kind permission.
[2] What is Active/Active?, *Availability Digest*; October 2006.

After we had tested several tools, we decided to use the open-source Nagios (www.nagios.org) infrastructure monitoring technology, which supports all systems except NonStop. As we had already expected, there was no Nagios client for NonStop available at that time. We therefore had no choice but to integrate the NonStop into the Nagios monitoring environment ourselves. This article explains what we did to ensure that Nagios can monitor NonStop in the same way that it monitors Windows, Linux and Unix servers.

The Nagios Monitoring System

Nagios is an open-source monitor that enables operations staff to identify and resolve IT infrastructure problems before they affect critical business processes. Nagios first became available in 1999 and has been continually enhanced by the worldwide Nagios community.

Nagios monitors systems, applications, services, and business processes to ensure that they are functioning properly. In the event of a failure, Nagios alerts technical staff so that they can begin remedial activities before business processes, end users, or customers are affected.

Integrating NonStop Servers with Nagios

In order to integrate NonStop servers with the Nagios monitor, the first step we tried was to create an SSH-session from the Nagios server to the NonStop system to execute commands and to return the command results to the Nagios server. This solution worked; however, it required a lot of configuration of the Nagios server, consumed a lot of resources and - worst of all – it was not reliable. Connecting the NonStop system to the Nagios server (or any other monitoring server without a NonStop client) thus required a completely new approach.

General Requirements

We identified the following requirements that we had to meet for the NonStop integration:

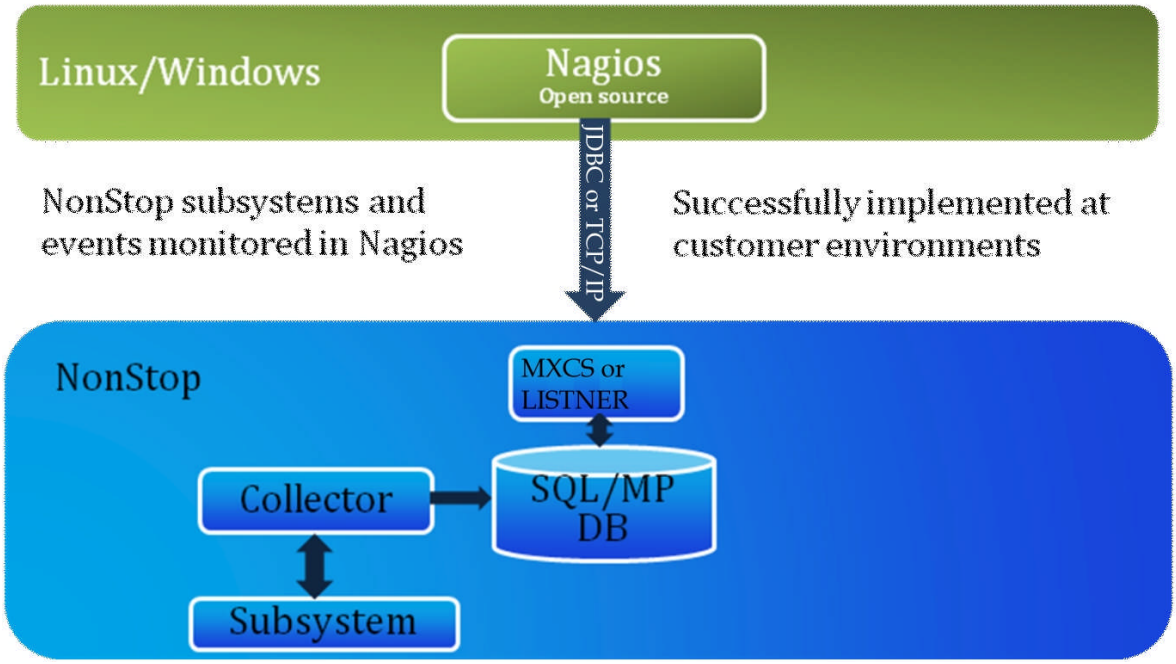
- Automatic configuration wherever possible.
- No privileged programs.
- Avoid NonStop specifics for the outside world.
- Handle problems automatically if possible.
- Reduce daily work for the system and database management staff.
- Provide statistical data.
- Improve documentation.

When we analyzed the problem, we came to the conclusion that the solution was to let the NonStop system perform all the monitoring tasks itself and to make the results available to the Nagios server. It was obvious that the best way to implement this approach was to use a database to store the NonStop monitoring results. As we wanted the monitoring to run within the NonStop Guardian operating system personality, we decided to use SQL/MP, the HP NonStop SQL database. This also enabled us to use NonStop MXCS so that ODBC or JDBC applications could access NonStop SQL/MP databases.

The next step was to define the monitoring functionality. We decided to create a specialized program for each monitored subsystem and started with the following subsystems:

- CPUs and processes
- Files
- Communication lines
- Pathway
- TMF

The sections below provide an overview of the main functions of our solution as it is implemented today.



Monitoring CPUs and Processes

The monitoring of CPUs and processes was the very first step. The program carrying out this function very soon got the nickname “Checker,” and so the program itself was named “Checker”.
“Checker” has many functions:

- Monitor CPU usage, queue-length and memory usage.
- Monitor number of PCBs (process control blocks) in use.
- Monitor existence of predefined processes.
- Find processes that use too many CPU cycles regarding exceptions.
- Reduce priority of those processes.
- If the busy process is a disc process: find the process and the file that caused the load.
- Build top-list of processes by CPU usage.

The process itself does not require much configuration. We maintain history tables containing all CPU data. That data is stored on an hourly and on a daily basis. With the help of that table, we are able to make predictions such as, for example, CPU usage during the days before Christmas in 2011.

All data (except searching for the file and process that are causing excessive disc-load) is retrieved without using MEASURE, the NonStop monitoring utility. We monitor to ensure that certain predefined processes exist. If a process is missing, we are able to restart the process automatically, provided a restart is configured. This functionality thus represents our own “Persistence Monitor”.

Monitoring Files

File monitoring is based on user-defined filesets like $DATA*.TABLE*.*. We check for full files; the criteria for this check can be specified in several steps down to a single file. Furthermore, we collect data about the last reload, last update statistics for SQL/MP and SQL/MX tables, and the mapping of ANSI-name to Guardian filename, including MP aliases. Finally we check key-sequenced files and tables periodically for necessary reloads and, if necessary, carry them out automatically. We check not only the free space but also the fragmentation. Here again, the criteria can be specified down to a single file.

Monitoring Communication Lines

When we started with the communication lines, one of our requirements was to avoid manual configuration of all lines. Instead, we let the software find all the configuration information. The result is a table containing all data for a line including SWAN-configuration (a SWAN device is a ServerNet Wide Area Network communication concentrator). The table allows us to find out all information about a specific line even if the user does not have any knowledge of NonStop-specific tools. Based on that database, we can monitor all configured lines without any manual intervention. In addition, we collect statistical data on an hourly basis, which gives us a good chance of finding long-term tendencies or lines with insufficient bandwidth.

Monitoring Pathway Servers

For Pathway monitoring, we use a similar approach. First, we find all running Pathway monitor processes; and then we retrieve all necessary configuration information from those processes. This allows us again to monitor Pathway servers based on automatically created tables. We also check whether too few or too many servers are running. In case there are too few static servers, we issue a “start server” command.

Monitoring TMF

The monitoring of TMF (Transaction Monitoring Facility) checks all TMF components, for example, audit trails, audit dumps and transactions. A very important function is the automatic backup of the TMF catalog; this automatic backup takes place as soon as an audit trail has been dumped. The resulting archive is stored on a system in another datacenter. This enables us to recover from a complete loss of a system - without the catalog there would be no chance to recover audited files and tables.

Monitoring Other Subsystems

Over the last two years, several other subsystems have been integrated as well. Now TCP/IP, Spooler, NetBatch, RDF (Remote Data Facility) and some other systems are monitored. And the list will continue to grow.

The procedure is very similar for all subsystems: Collect all necessary information automatically and then monitor the system based on the information gathered. Of course, it is possible to define exceptions, for example a line that is defined but not used. We furthermore have a defined set of parameters. All these parameters are stored in a central parameter table and all processes are able to read this table online by command at any given time.

Monitoring EMS Messages

A real challenge was to find a solution that would enable the program to handle EMS (Event Management Service) messages. Every EMS message has a unique description, for example “Application timeout on line $X25”. The message is defined by up to five text-portions that have to be contained either in the message or must not be contained in the message. In the example, the original EMS message would have to contain “application timeout” and “$X25” in whichever order.

Every EMS message that the program reads is checked, and the messages that do not match the search criteria are dropped. By setting the parameters accordingly, we can define how many occurrences of an

EMS message within a defined time period should lead to a warning. It is also possible to define that certain EMS messages have to be processed manually. Such messages receive a case number and can only be marked as “resolved” manually.

This is the only program that does not have a default configuration.

Message Collector

All messages created by our NonStop monitoring facilities are collected by a program we call the “Message Collector”. Every message is acknowledged by the Message Collector. All messages that are still active and all information about the current CPU statistics are included in the Messages Table.

If a problem has been solved, this fact is also documented, and a “solved” message is created. Furthermore, all messages are written to an EMS collector and can be retrieved using a specific filter. To produce the messages, a template is used and the keywords within the template are replaced with the information of the reported event. The template is stored in a table. As the language is part of the key, this program can produce messages in any language if the associated templates are available.

Here is an example for a files message:

Message FILES 2224W
#FILENAME is #PERCENT full, #ALLOCATED of #MAXIMUM extents allocated, EOF #EOF, max. #MAX-EOF, file-format #FORMAT, reload checked #RELOADCHECK

The keywords starting with # are replaced by their values and the rest of the template is just copied to the message:

FILES 2224W
$DATA77.TABLE.DATA is 75.7% full, 711 of 940 extents allocated, EOF 14638125056, max. 19333120000, file-format 2, reload checked 2011-09-06:20:03

Nagios Server

The last and smallest part of the solution is the interface to the Nagios server. The Nagios server prompts the NonStop systems for data via TCP/IP. If a request comes in, the LISTNER starts a process that retrieves the requested information from the Messages Table and sends it to the Nagios server. The Nagios server furthermore has a user exit to handle user-specific additional information. We use this exit to retrieve and send the current status of our authorization application.

Thanks to the simplicity of this interface, it would be very easy to connect to another monitoring server. All monitoring servers supporting ODBC or JDBC could access the Messages Table directly.

Nagios GUI

This screenshot below shows the overview of our NonStop production systems. The upper half of the screenshot shows the current CPU usage and the summary status of all the X.25 and SNA lines. Below that, we see our monitored subsystems for both systems. In the middle you can see the application data delivered by the user exit of the Nagios interface.

In this example, everything except BAT is green; this indicates that something is wrong with a batch job and that the OS release on one system is older than the one on the other system. A click on BAT gives detailed information about the problem.

The Benefits of Our Effort

Naturally, the creation of our own monitoring system has not been easy or simple. We had to read many manuals and call the support centers on various occasions because of missing manuals or missing information in the manuals. Additionally, we posted some questions in the Google group and got some useful replies. However, it has been and still is a really interesting project and there are a couple of new and interesting ideas - like a help database (what can I do if a special message occurs) - waiting to be investigated and implemented.

The monitoring solution has provided our first level team with a lot of information, and it is saving a lot of time with regard to our daily work. Since we started using this monitoring solution, the daily work for system and database management has been reduced significantly. Manual reloads in particular have become extremely rare.

We collect a lot of statistical data that allows creating reliable predictions for critical dates like Christmas or Easter. We also store a significant amount of information about the system and application configuration in order to improve documentation. An example of this is the table containing the “system globals”. The table is maintained automatically and contains all general system information (system name, system number, serial number, OS release, SYSnn) including the history for all those values. We therefore have complete documentation about release changes. The table is designed to contain the data of more than one system, which enables us to access the data of all systems from all systems. This means that if a system fails unexpectedly, we do not have to access any documentation to find the current SYSnn. A simple SQL-select on one of the other systems provides that information within seconds.

The monitoring itself does not dependent on a specific server, for example, Nagios. We use Nagios as a frontend and GUI. All information is available from the NonStop directly either by using simple SQL

queries or via EMS. This is especially helpful in a crisis - all information about problems are available directly from NonStop.

During the course of our work, we were happy to discover advantages that we had not expected. The process monitoring in connection with the restart tool made it possible to start the system and all applications without any manual action and, even more importantly, without the use of SUPER.SUPER privileges. The restart tool is started during the cold-load and initiates all necessary processes. The sequence of the restarts is defined by priority. We tested this procedure with our development system and the tests were very successful. So far, we are unable to test it on another system because there has not been a cold-load since then.

The advantages at a glance:

- Easy installation, takes only 15 minutes.
- Very little manual configuration.
- Only about 0.25% CPU usage on an NS16004
- The first level support needs only very limited knowledge on NonStop.

So to conclude: Yes, we would do it again!