WORM AND PEER TO PEER TOOLS FOR DISTRIBUTION AND MANAGEMENT OF ATLAS SW ON TDAQ COMPUTER CLUSTERS

H. Garitaonandia, IFAE, Barcelona, Spain
H. Zobernig, University of Wisconsin, Madison, USA

Abstract

ATLAS TDAQ software, with six GBytes per release, will be installed locally in about two thousand machines in the final system. Already during the development phase, it is tested and debugged in various Linux clusters of different sizes and network topologies. SW managing tools are not always available. For the distribution of the software across the network in such cases two technologies have been tested: virus or worm technology with fixed distribution paths, and adaptive distribution. The first one has been implemented with a worm named Nile. It is a utility to launch cryptographic connections in a mixture of parallel and cascaded modes, in order to synchronize software repositories incrementally or to execute commands. A system administrator configures, in a single file, the routes for the propagation. Therefore it achieves scalable delivery, as well as being efficiently adapted to the network. The installation of Nile is trivial, since it is able to replicate itself to other computers memory. Moreover, robustness is issued by the use of routing and monitoring protocols together with an adaptive runtime algorithm to compensate for broken paths. The other technology, adaptive distribution, is implemented with peer to peer protocols, or P2P. In this solution, a node interested in a file acts as both client and server for small pieces of the file. The strength of the P2P comes from the adaptive algorithm that is run in every peer. Its goal is to maximize the peer's own throughput, and the overall throughput of the network. Hence the network resources are used efficiently, with no configuration effort. The selected P2P tool is BitTorrent. This paper overviews the architecture of Nile and BitTorrent and their applications, describes tests performed in CERN clusters of 12 to 600 nodes with both technologies, and compares the benefits of each.

INTRODUCTION

System administrators and cluster software developers often have to face the problem of installing programs or patches in farms with no suitable tool available. It may be due to incompatibilities between the packaging systems,or a restrictive configuration of the installer program, etc.

Moreover, there must be an intermediate approach between more featured tools like Quattor[1], or similar commercial tools, and a simple parallel copy to every computer. This approach should be ready to deal with heavy installations in heterogeneous external clusters, of different sizes and network topologies. It should have an straightforward configuration so that installation systems can be setup quickly, and still be performant.

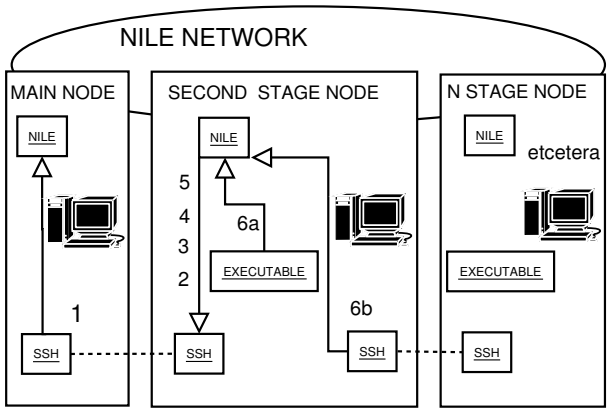

Figure 1: Nile process hierarchy for execution mode (numbers indicate sequential steps)

For this purpose two lightweight systems have been developed: one with fixed distribution paths based on worm technology, a program named Nile[2], and another one adaptive, based on P2P technology. Neither of them pretend to be substitutes of tools like Quattor, they are complementary instead.

Nile's design is such, that it was easily extended with the capability of launching some executables. This feature opens new possibilities.

NILE

Nile is a worm that executes commands in a given list of hosts or copies files to them. In order to reach all the machines, it creates a hierarchical control network via worm propagation, so with only one computer hosting Nile, the worm is able to reach all the others without accessing their hard disk.

It can create any propagation tree as defined by the user in a configuration file, or decide itself how to arrive to those nodes on the basis of some configured restrictions, such as maximum number of branches per node. In any of these cases, if a path is found to be broken or a connection is lost at some point, the last node in contact with the main node rearranges the following paths in order to arrive to all the remaining nodes. The branch that looses contact with the root node kills itself abruptly, and a new branch is generated by the previous node.

Nile has three types of shutdown: *upon exit*, *requested*

by user and *abrupt*. They make use of Nile protocol and POSIX interprocess communication mechanisms to guarantee that every reachable node is reached once, and that the centralized logs report information for every node, even if it was not up or there was a network error.

Nile is inspired in Distribulator[3], and Rgang[4], but written from scratch by author H. G. in order to improve functionalities. It also borrows a couple of ideas from Metasploit Project[5].

Nile in Execute Mode

When execute mode is set, it can either launch a shell command in a given list of hosts, upload a perl script to their memory and execute it, or execute different commands on different hosts.

Figure 1 shows the job generation procedure. In step 1, the main copy of Nile launches N SSH[6] parallel processes to N computers. For each SSH client process, an SSH server process is generated in the next computer. This server process (step 2) runs a perl command interpreter with a small command specified when calling the SSH client process in the main node. SSH server process redirects the server end of the channel to that perl command. Then (step 3) the perl command starts reading and saving to a virtual buffer what the SSH client is sending: a fresh copy of Nile. Finally (step 4) the command passes the execution control to the start of that buffer, and (step 5) there is a copy of Nile running on every node of the second stage.

This copy of Nile tells the one in the main node, via the established SSH connection, that it is ready. Then receives a packet with the routing information for the next nodes and the executable (bash command, perl script...). It creates the local executable process in a *sandbox*, and if there are any nodes left pending from the branch, it generates the packets for these nodes, and it launches the SSH processes to them.

When the local executable dies, the Nile process in that computer packs its output and its return code, and sends them to the previous Nile. It also forwards all the packets sent by the slave Nile processes in other machines. The root Nile unpacks the information and presents it to the user.

Though the current number of parallel processes can be limited at any time and Nile can be configured to be executed in only one stage, the tree like execution has an advantage over this flat configuration, and even more for programs with short execution time. The advantage comes from a more simultaneous execution of the long SSH handshake T_{SSH}. In a 600 machine network, with a two stage configuration we have $2(T_{\text{SSH}} + T_{NILE})$, and only 25 $(= \log_2 600)$ processes are executed in a single machine. With a flat configuration, limiting the number of parallel processes to 25 we have about $\frac{600}{25} T_{\text{SSH}} = 24 T_{\text{SSH}}$. Note that T_{NILE} is usually equal to T_{SSH}.

Nile in Copy Mode

In copy mode Nile is able to synchronize incrementally many repositories with the main one. The use of RSYNC[7] over SSH as synchronization protocol makes possible to transfer only modified and absent files.

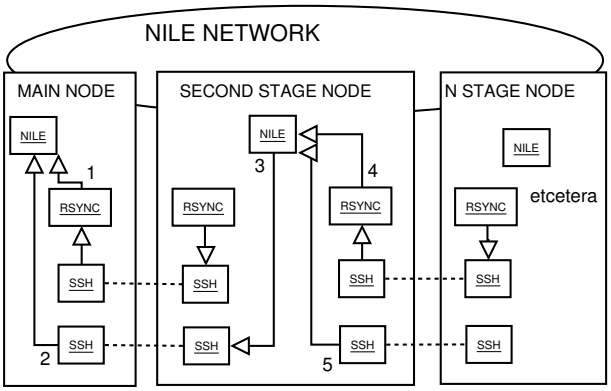

Figure 2: Nile process hierarchy for copy mode

The procedure (see figure 2) is very similar to the one in execute mode. With the only difference that before jumping to a computer it first has to be correctly synchronized.

Another option is to use RSYNC on top of TCP, so that the limiting factor of a synchronization is not the SSH CPU consumption, but the network. When using one RSYNC job over SSH in a Gbit Ethernet and with a hyper threaded dual processor Xeon at 3.06 GHz, one processor reaches the 100% usage due to the SSH encryption, and the throughput drops down to 80Mbps. If RSYNC jobs are run over TCP in the same HW, with no compression and with synchronization at file level (not at file piece level [8]), 800Mbps throughput is reached at low CPU cost.

This way, in a network topology like the one in the figure 3, where all the switches have the same nominal capacity, the expected total transfer time can be modeled by:

$$T_T = T_{STAGE1} + T_{STAGE2} = (N + M - 1)\frac{F}{R}$$

And the throughput by:

$$R_{EFF} = \frac{FNM}{T_T} = R\frac{NM}{N+M-1}$$

where R is the rate of any switch, F the file size, and with Nile configured in two stages to synchronize first a machine in each of the leaf switches, and then from that machine synchronize the rest of the machines in the leaf switch.

Otherwise, if the file is transfered in parallel from the main repository to all the leaf nodes at the same time, the expected transfer time is:

$$T_T = NM\frac{F}{R}$$

BITTORRENT

BitTorrent[9] is a P2P file sharing software which adapts to the network topology at switch level for overall efficiency. This is possible thanks to its distributed implemen-

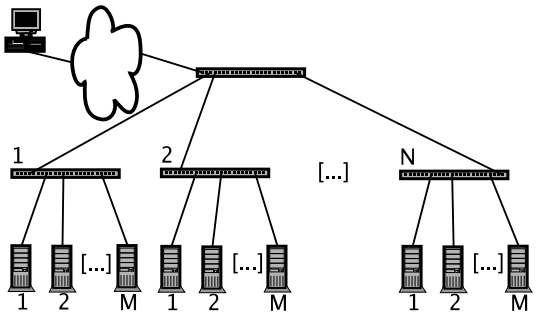

Figure 3: Network topology

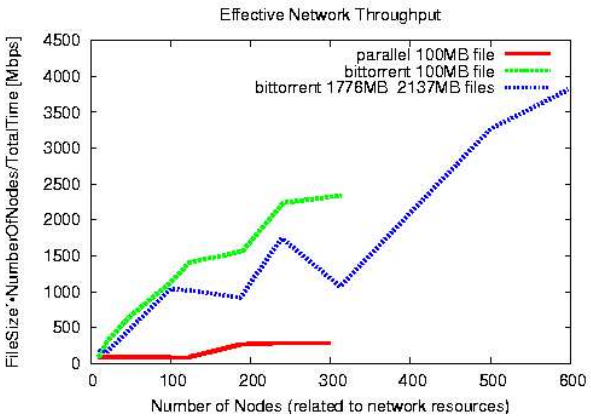

Figure 4: Network throughput of BitTorrent compared to parallel copy in LST. One measurement per point.

tation. It is good at distributing large files, but no incremental synchronization is possible. For incremental P2P synchronization see PeerFS[10].

Architecture

These are the elements that build the architecture of a BitTorrent network:

- Many equal peers interested in the same object file. A special case of the peer is the seeder, a peer with a full copy of the object file. The executable that provides the peer's software is btdownloadheadless.py.
- The tracker, a daemon that helps peers find each other. It also lets them know where the pieces that conform the object file are. It uses HTTP as transport protocol.
- A metainformation file that is accessible via an URL by all the peers. This file contains hash information about the object file and a pointer to the tracker.
- An adaptive algorithm that runs on every peer. It is responsible for opening and choking connections with other peers to maximize the individual throughput.

The peers and at least one seeder access the metainformation file via URL. They extract the information about the pieces of the object file, and the location of the tracker. They connect to the tracker so that the tracker knows about them, and so that the tracker can provide them with information about the others. When a peer gets information about other peers location, and about the pieces of the object file that these other peers hold, it starts downloading from the other peers. And it also uploads upon request its own pieces of the object file to others. When a peer finds out that it is not getting much throughput from another one, it closes that connection and searches, with the help of the tracker a better one. Hence, an apparently paradoxical effect of BitTorrent networks is that the more peers it has, the more efficiently resources are used. The correct delivery of the object file is guarantied by the SHA1 integrity check performed on every piece of the object file, as well as in the whole file.

Start, Monitor and Stop BitTorrent Network

For the purpose of starting the BitTorrent network, a set of scripts based on Nile were developed. Nile launches the 600 btdownloadheadless.py binaries of the peers redirecting their output to a local file. Then sequential Nile executions parse the continuous output of the peers, and when all the peers have finished, either successfully or with error, a third run of Nile stops the network.

CURRENT APPLICATIONS AND TESTS

SW Distribution in Large Scale Tests at CERN

In June 2005, a 2 GByte file containing a compressed monolithic installation of ATLAS Trigger DAQ had to be deployed on 600 machines, for large scale tests (see [11]). The official distribution tool of the Information Technologies department, that manages these computers, was Quattor. Quattor was configured so that it could only install packages in RPM format, which was different than the one used for the monolithic ATLAS SW. Moreover, these 600 machines were spread over different physical locations, conforming a virtual cluster, so there was no clear knowledge about the network topology.

A P2P solution was ideal for this task. As it can be observed in figure 4, BitTorrent performed better than parallel copy. Nile's full functionality was not yet available then.

For the parallel copy, two different seeders were chosen, one in front of a 100Mbit switch (up to 123 machines) and a 200Mbit router for the rest of the machines.

$$R_{EFF} = \frac{NMF}{T_T} = R$$

The equation above shows that when increasing the number of machines and also the network resources, the whole throughput of the network is limited by the first switch, so the transfer time grows linearly with the number of machines. On the other hand, with BitTorrent, increasing the network resources means gaining overall throughput. An

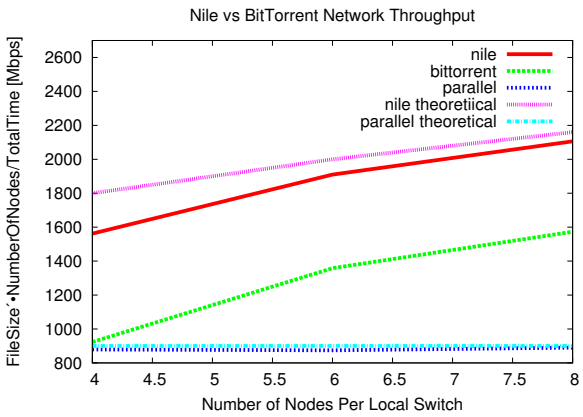

Figure 5: Network throughput of Nile, BitTorrent, and parallel copy (emulation of a simple Quattor configuration). Four measurements per point.

early version of Nile was used during Large Scale Tests to launch BitTorrent network.

SW Distribution with Known Network Topology at PreSeries

PreSeries is a small scale system which represents 10% of the final ATLAS TDAQ (see [12]). The topology for the control network is the one of figure 3, with N=4 and M_i=$\{$8,12,30,15$\}$. Until Quattor will be made available, ATLAS SW is synchronized to local disks by Nile. The system has profited from its incremental synchronization when small patches had to be applied. Also during commissioning of the network, Nile was used to partition the hard disks, and other administrative tasks.

For the performance comparison tests, due to practical restrictions, networks with only N=3 and M=$\{$4,6,8$\}$ were used.

Figure 5 shows that Nile performed better than BitTorrent, and also better than parallel copy. Its performance was close to the expected:

$$R_{EFF} = R\frac{NM}{N+M-1} = 2700\frac{M}{M+2}$$

The parallel copy can be interpreted as an emulation of the simplest Quattor configuration, with only one SW repository and a modified client. But more complex configurations of Quattor would lead to throughputs close to 2700Mbps.

Nile as Transport for Other Tools: TDAQ Diagnostic Tools

Nile can be used as transport mechanism for executables by other tools thanks to its reliability, its centralized logs, its well defined interface, and some useful features like *different command on different computers*. A set of tools, currently under development, is being built on top Nile. Their objective is to provide, in a fast way: Unix resource monitoring, TDAQ infrastructure monitoring and diagnostic, deallocation of resources, etc.

CONCLUSIONS AND FURTHER WORK

When no SW package distribution tool is available we need an easily configurable and lightweight tool as a temporary solution. Two technologies are appropriate for this: P2P and worm. The P2P BitTorrent, thanks to its adaptability is appropriate for unknown network topologies as demonstrated in LST at CERN. The worm Nile performs better for low loaded and more symmetrical networks like the one in PreSeries at CERN. Moreover it has the functionality of synchronizing incrementally at file and file piece level, thanks to the underlaying RSYNC.

Nile alone, or the combination of both tools provide a reliable software distribution mechanism for future large scale tests in clusters outside CERN.

It would be an interesting issue to integrate Quattor with BitTorrent, that are compatible because they are both pull (rather than push) oriented. Or even invest some time and resources in studying and developing a distributed file system on top of P2P that fits ATLAS and LCG needs.

Nile, thanks to its reliable and fast transport mechanism for executables, is appropriate to build tools on top of it, like TDAQ diagnostic tools. It is also useful in other administrative tasks during commissioning of a network.

ACKNOWLEDGMENT

The authors would like to thank their colleague Gokhan Unel who was the first user of Nile. We would also like to thank TDAQ Sysadmins group, and the ATLAS TDAQ Collaboration for their support.

REFERENCES

[1] http://quattor.web.cern.ch/quattor/

[2] http://nile.ifae.es

[3] http://sourceforge.net/projects/distribulator/

[4] http://fermitools.fnal.gov/abstracts/rgang/

[5] http://www.metasploit.com

[6] http://www.openssh.com/

[7] http://www.samba.org/rsync/documentation.html

[8] Andrew Tridgell. Efficient Algorithms for Sorting and Synchronization. http://samba.org/tridge/phd_thesis.pdf, Feb 1999.

[9] Bram Cohen. Incentives build robustness in bittorrent. http://www.bittorrent.com/bittorrentecon.pdf, May 2003.

[10] http://www.radiantdata.com/English/Products

[11] Testing on a large scale: Running the Atlas Data Acquisition and High Level Trigger software on 700 pc nodes. CHEP, Feb 2006

[12] Studies with the ATLAS Trigger & Data Acquisition "preseries" setup. CHEP, Feb 2006.