JARLE BJØRGEENGEN

Puppet and Cfengine compared: time and resource consumption

Jarle Bjørgeengen has been working with UNIXes, storage, and HA clusters for about a decade. He is now a master's student at Oslo University College, and works for the UNIX group of the central IT department (USIT) of the University of Oslo.

jarle.bjorgeengen@usit.uio.no

DURING THE MASTER'S PROGRAM AT Oslo University College, I was required to define and carry out a set of scientific experiments. As both graduate student and member of a project evaluating configuration management tools for the University of Oslo's IT department [9], I was looking for an experiment that would serve both of these roles. The project was about evaluating Cfengine [3] and Puppet [5] against our existing script-based solution.

My experiment needed to fulfill the criteria of scientific measurability and bringing a new and valuable approach to the decision-making process. I chose to compare time and resource consumption in Puppet and Cfengine 3 tools when carrying out identical configuration checks and actions.

This would be useful information to have for a tool to be used on a large scale, since the time and resources used limit the scope of managed configuration during a certain period. Time usage for the state compliance verification process is of particular interest, since it will affect the frequency and/or scope of the configuration verified.

Equipment and Tool Setup

The experiment was run on a PC with the following specifications:

- CentOS 5.2 , 2.6.18-92.1.22.el i686
- MSI MS-6380E (VIA KT333 based) motherboard
- 1024 MB RAM
- AMD Athlon XP2200 (1.8GHz / 266MHz FSB) 256KB cache
- Quantum fireball 7200 rpm / 30GB / 58169 Cyl / 16 Head / 63 sectors

In both tools it is the configuration agent that does the job of converging to desired state. In Puppet a server component is mandatory, since it is the server that provides the agent with its configuration. The latest stable version of Puppet at the time the experiment was done was version 0.24.7 and that is the version used in the experiment.

Cfengine's configuration agent is independent of a server component, but can be configured to be used with a server component if desired. Cfengine version 3.0.1b3 was downloaded and was compiled according to the installation part of the reference manual [2].

Installing Puppet is easy on CentOS 5 using the EPEL [1] yum repository.

```
# rpm -Uvh http://download.fedora.redhat.com/pub/epel/5/i386/epel-release-5-3.noarch.rpm
# yum install puppet
# yum install puppet-server
# service puppetmaster start
# chkconfig puppetmaster on
```

This also automatically resolves and installs the dependencies needed for Puppet to work (ruby-libs, ruby, facter, ruby-shadow, augeas-libs, ruby-augeas).

Installing Cfengine 3 took a bit more effort:

Dependencies:

```
# yum install db4-devel
# yum install byacc
# yum install openssl-devel
# yum install flex
# yum install gcc
```

Cfengine:

```
# wget  http://www.cfengine.org/downloads/cfengine-3.0.1b3.tar.gz
# tar zxvf cfengine-3.0.1b3.tar.gz
# cd cfengine-3.0.1b3
# ./configure && make && make install && cp /usr/local/sbin/cf-* /var/cfengine/bin/
```

Methodology

The workload chosen for the measurements reflects comparable activities of a configuration management tool. There are three main categories of work:

- File permissions
- File contents
- /etc/hosts entries

The host entries measurement differs a little bit from the file permission and file content workloads. This is because, in Puppet, host entries means a special type of resource, which ends up as file edits to the /etc/hosts file in the end. In Cfengine this is just one form of file-editing operation directly in the configuration language. This difference might make the /etc/hosts entries workload less directly comparable than the file permission and content workloads.

Each type of work was measured in two ways:

1. With a known deviation from the tool's configuration applied up front. Hence the tool will need to converge the deviation to compliance.
2. With the tool's configuration applied up front. Hence the tool will do verification only.

This expands to six measurements for each tool, and each measurement was repeated 40 times. All measurements and application of preconditions for each measurement were done in a shell script which ran the same measurements 40 times.

All workload combinations measured are summarized in the following table:

No.	Type	Converge or Verify	Tool	configfile
1	Permissions	Converge	cf3	cf3-perms.cf
2	Permissions	Converge	puppet	puppet-perms.cf
3	Content	Converge	cf3	cf3-content.cf
4	Content	Converge	puppet	puppet-content.cf
5	Hosts records	Converge	cf3	cf3-hosts.cf
6	Hosts records	Converge	puppet	puppet-hosts.cf
7	Permissions	Verify	cf3	cf3-perms.cf
8	Permissions	Verify	puppet	puppet-perms.cf
9	Content	Verify	cf3	cf3-content.cf
10	Content	Verify	puppet	puppet-content.cf
11	Hosts records	Verify	cf3	cf3-hosts.cf
12	Hosts records	Verify	puppet	puppet-hosts.cf

Each amount of work was scaled up such that it was possible to actually measure some resource and time consumption for both tools. For the file permissions and file content tests, a file tree with known content and permissions was created under a test directory */var/tmp/file_tests*.

The test file tree was made each time as follows:

```
TESTDIR=/var/tmp/file_tests
if [[ -d $TESTDIR ]]
then
   rm -rf $TESTDIR
fi
for i in `seq 1 10`
do
   mkdir -p $TESTDIR/dir_$i
   for j in `seq 1 10`
   do
      echo "Some testline" > $TESTDIR/dir_$i/file_$j
   done
done
```

Run as root, this creates directories with ownership root:root and mode 755 and files with ownership root:root and mode 644.

For the file permissions, the tools' work was to converge from the default permissions to ownership root:bin and mode 755 for all files and directories.

Here's the Puppet configuration for applying file permissions (puppet-perms.cf):

```
class fix_perms {
   file { "/var/tmp/file_tests":
      owner => "root",
      group => "bin",
      mode  => 0755,
      recurse => inf,
      backup => false,
   }
}
```

```
node localhost {
  include fix_perms
}
```

And here's the Cfengine configuration for applying file permissions (cf3-perms.cf):

```
body common control
{
bundlesequence  => { "perms" };
}

bundle agent perms
{

files:
  "/var/tmp/file_tests/"
  pathtype => "literal",
  perms => passthrough("0755","root","bin"),
  depth_search => recurse("inf");
}

body depth_search recurse(d)
{
  depth => "$(d)";
}

body perms passthrough(m,o,g)
{
  mode  => "$(m)";
  owners  => { "$(o)" };
  groups  => { "$(g)" } ;
}
```

For file content, the tools' work was to ensure some particular content in the same files.

Here's the Puppet configuration for applying file content (puppet-content.cf):

```
class fix_contents {
  file { "/var/tmp/file_tests":
     content  => "Trallala\n",
     recurse => inf,
     backup => false,
  }
}

node localhost {
  include fix_contents
}
```

And here's the Cfengine configuration for applying file content (cf3-content.cf):

```
body common control
{
bundlesequence  => { "content" };
}

bundle agent content
{

files:
```

```
    "/var/tmp/file_tests/.*/.*"
    edit_defaults => no_backup,
    edit_line => en_liten_trall;
}

bundle edit_line en_liten_trall
{
    delete_lines:
    ".*";
    insert_lines:  "Trallala";
}

body edit_defaults no_backup
{
    edit_backup => "false";
}
```

For host entries, the tools' work was to ensure that all specified host-to-IP mappings were present in /etc/hosts.

Here's the Puppet configuration for applying host entries (puppet-hosts.cf):

```
class my_hosts {
    host { "private1.localdomain.com":
    ip => "10.0.0.1",
    ensure => present,
}

host { "private2.localdomain.com":
    ip => "10.0.0.2",
    ensure => present,
}

(...)

host { "private254.localdomain.com":
    ip => "10.0.0.254",
    ensure => present,
    }
}

node localhost {
    include my_hosts
}
```

And here's the Cfengine configuration for applying host entries (cf3-hosts.cf):

```
body common control
{
    bundlesequence  => { "hosts" };
}

bundle agent hosts
{
        vars:

    "my_hosts" slist => {
        "10.0.0.1 private1.localdomain.com",

        (...)

    "10.0.0.253 private253.localdomain.com",
    "10.0.0.254 private254.localdomain.com"
```

```
};

files:

   "/etc/hosts"
   edit_defaults => no_backup,
   edit_line => host_ensure("@(hosts.my_hosts)");
}

bundle edit_line host_ensure(record)
{
        insert_lines:

"$(record)";

}

body edit_defaults no_backup
{
   edit_backup => "false";
}
```

When measuring verification time, the script converged configuration of the tool before taking the measurements. This way it is unlikely that the precondition is anything other than the desired state in the tool configuration; hence the measured values when doing this are verification only.

Application of the configurations with Puppet was done like this:

```
/usr/sbin/puppetd --no-daemonize --onetime
```

Application of the configurations with Cfengine was done like this:

```
/var/cfengine/bin/cf-agent --no-lock
```

To have as close to equal starting points as possible for the tests, all block device buffers were dropped before each test by using:

```
sysctl -w vm.dropcaches = 3
```

The Scientific Method

When measuring alternatives there will be uncertainty in the measurements. The uncertainty is defined as error, or noise. The total error was quantified using repeated measurements and statistical methods for finding the confidence intervals of the differences between alternatives. Confidence intervals of the differences comes with a probability of the true value being inside the interval. For the confidence interval to have any utility, the probability that the true value is inside the interval must be high. Higher probability widens the confidence interval, and vice versa. A commonly chosen value of this probability is 0.95.

If confidence intervals of the two alternatives overlap, it is impossible to say that the difference is not caused by random fluctuations. If they don't overlap, there is no evidence to suggest that there is *not* a statistically significant difference. The chosen probability quantifies the certainty of being right in assuming there is a true difference [10, p. 43].

Plotting the values shows random variations, but the true standard deviation of the underlying population is not known. The student's *t* distribution takes this into account and is commonly used for detecting statistically significant differences between two alternatives [8]. The student's *t* distribution, or more precisely the *t*-test, was used to identify any statistically significant differences in the experiment results.

The tool "gnu time" [4] (version 1.7 release 27.2.2) was used for measuring time and resource consumption of each operation. For each measurement, three metrics were logged:

- Total time used
- Number of CPU seconds (system + user)
- Number of involuntary context switches

The values were logged in semicolon-separated files, one for each $measurement-$tool containing 40 lines of data. Each column of measurements then represents one metric (time, CPU, or CSWITCH) measurement repeated 40 times. These vectors were used for calculating sample means and confidence intervals using the free statistic program R [6].

For all 12x3 metrics, the *t*-test functions of *R* were used to compare pairs of vectors of 40 numbers, each representing the measured values of each tool, respectively. The outcome of each comparison is the sample mean of the difference between vectors and its confidence interval given a certain probability that the true value is within the interval.

The probability used for the tests was 0.99, meaning there is 99% probability that the true value of the sample mean of the differences is within the confidence intervals produced by the *t*-tests.

All *t*-tests were done as follows:

```
diff = t-test(puppet_vector,cfengine_vector,conf.level = 0.99)
```

and the return values are fetched out as follows in *R*:

```
c(diff$estimate[1],diff$estimate[2],diff$conf.int[1],diff$estimate[1] -
diff$estimate[2] , diff$conf.int[2])
```

The five values produced from the two statements above correspond to columns 3–7 in the results table.

Results

The output of the eighteen *t*-tests in *R* is summarized in the following table.

Table legend:

1. Type of workload: Permission/Content/Host Converge/Verify
2. Resource/time measurement
3. Sample mean value for Puppet
4. Sample mean value for Cfengine
5. Start of the confidence interval of the sample mean difference (C1)
6. Sample mean difference
7. End of the confidence interval of the sample mean difference (C2)

Workload	Measurement	Puppet mean	Cf. mean	C1	Mean difference	C2
Permissions conv.	Execution time	14.80s	1.18s	13.54s	13.63s	13.72s
Content conv.e	Execution time	14.85s	1.43s	13.24s	13.42s	13.6s
Hosts conv.	Execution time	28.44s	1.21s	26.19s	27.23s	28.27s
Permissions ver.	Execution time	14.28s	1.25s	12.82s	13.04s	13.25s
Content ver.	Execution time	14.32s	1.23s	12.92s	13.09s	13.27s
Hosts ver.	Execution time	21.52s	1.11s	20.30s	20.41s	20.53s
Permissions conv.	cpu seconds	11.03s	0.24s	10.77s	10.79s	10.81s
Content conv.	cpu seconds	11.03s	0.36s	10.64s	10.67s	10.69s
Hosts conv.	cpu seconds	10.50s	0.32s	10.16s	10.18s	10.19s
Permissions ver.	cpu seconds	11.04s	0.23s	10.78s	10.8s	10.8s
Content ver.	cpu seconds	11.03s	0.34s	10.67s	10.69s	10.72s
Hosts ver.	cpu seconds	17.97s	0.32s	17.61s	17.65s	17.68s
Permissions conv.	forced cswitch	1861	150	1595	1711	1827
Content conv.	forced cswitch	1918	159	1644	1759	1874
Hosts conv.	forced cswitch	3194	155	2929	3039	3148
Permissions ver.	forced cswitch	1933	151	1675	1782	1889
Content ver.	forced cswitch	1967	160	1708	1808	1907
Hosts ver.	forced cswitch	3186	159	2913	3027	3142

The following graphs are produced by the package Sciplot [7] in R. The error bars show the 99% confidence interval of each sample mean.

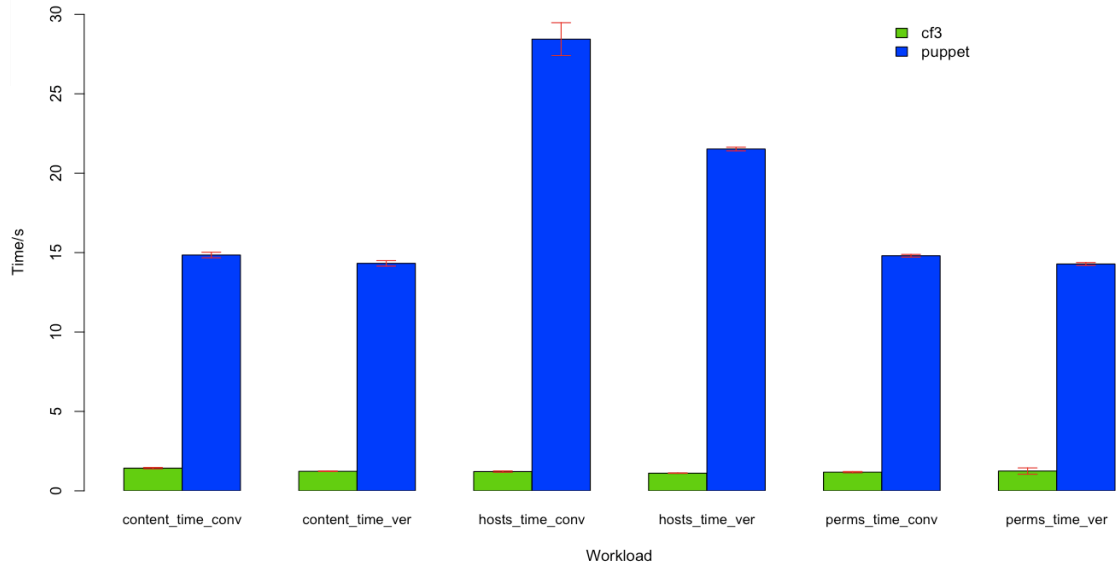

FIGURE 1: TIME USAGE FOR THE SIX TASKS

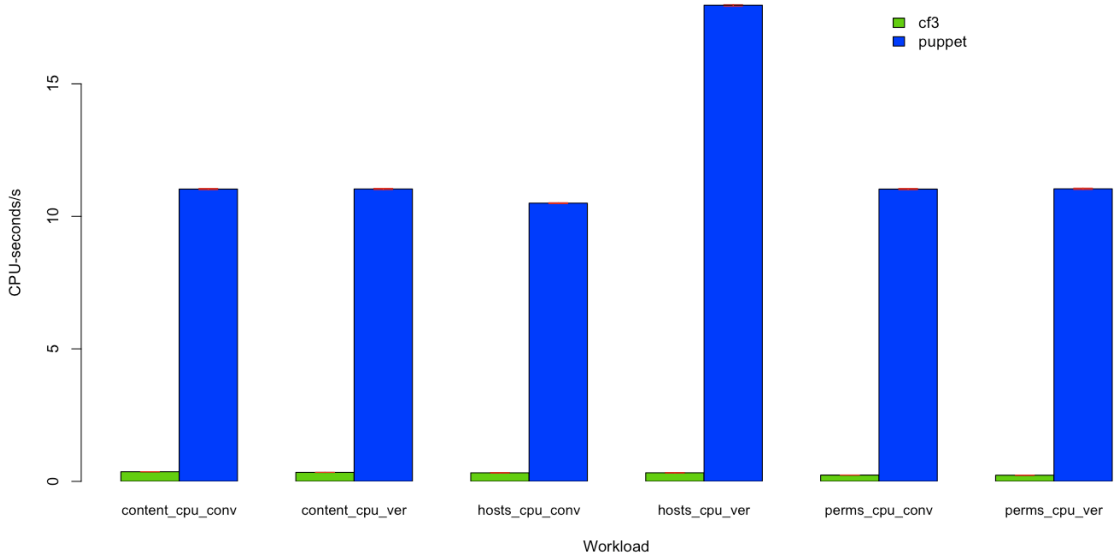

FIGURE 2: CPU SECONDS USED

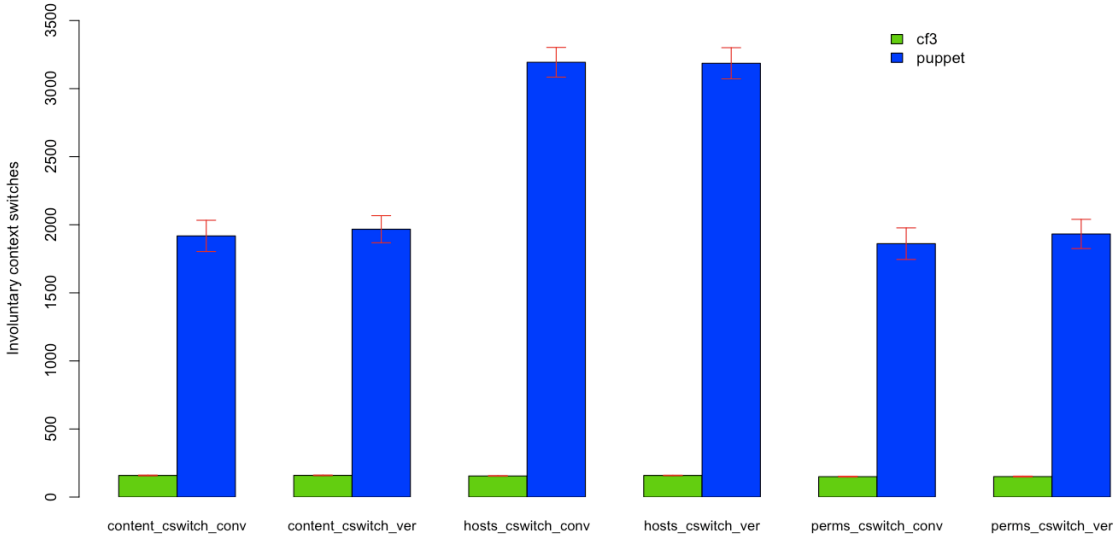

FIGURE 3: NUMBER OF INVOLUNTARY CONTEXT SWITCHES

Conclusion

The results show that Puppet uses considerably more time and resources than Cfengine3 for all the measurements included in the experiment. Time and resource usage is important, particularly in the verification phase. Verification is done every time the agent runs, regardless of compliance. The maximum frequency of verifications will be affected by time usage for each verification. The scope of the verified configuration in the experiment is small compared to what can be the case in real production environments. Clearly, time usage of verifications will limit the frequency of verifications when the scope increases.

Generally it is also desirable to have low resource consumption on administrative processes that run regularly, both from an environmental and from a capacity point of view.

The experiment shows that usage of Puppet involves a major tradeoff with respect to time and resource consumption compared to Cfengine3 for the operations that were measured.

Of course, there are many factors to consider when choosing configuration management tools. The differences of time and resource consumption might be ignorable to some. The results of this experiment serve as a supplement, broadening the understanding of the differences between these two popular configuration management tools.

REFERENCES

[1] http://fedoraproject.org/wiki/EPEL.

[2] http://www.cfengine.org/manuals/cf3-reference.html.

[3] http://cfengine.com/pages/whatIsCfengine.

[4] http://www.gnu.org/software/time/.

[5] http://reductivelabs.com/products/puppet/.

[6] http://www.r-project.org.

[7] http://cran.r-project.org/web/packages/sciplot/index.html.

[8] http://en.wikipedia.org/wiki/Student's_t-distribution.

[9] http://www.usit.uio.no.

[10] D.J. Lilja, *Measuring Computer Performance: A Practitioner's Guide* (Cambridge University Press, 2000).