A control theory perspective on configuration management and cfengine

Mark Burgess
Oslo University College, Norway

Abstract— Cfengine is an autonomous agent for the configuration of Unix-like operating systems. It works by implementing a hybrid feedback loop, with both disrcete and continuous elements.

Keywords— **Control theory, configuration management.**

I. Introduction

Configuration management is the business of enforcing predictable patterns of resource deployment and maintenance in computer systems. In the past, configuration management has often been viewed as a static, one-off task to be performed at the start of a system's history. Increasingly however, researchers are coming to realize that the unpredictable nature of computers in their environments mandates a dynamical feedback process.

Cfengine is a widely used configuration management tool and an on-going research project, looking at distributed configuration management. Since its inception in 1993, the cfengine software has been adopted by a broad range of users from small businesses to huge organizations[1]. It is currently running on an estimated million nodes around the world. Cfengine may be described as a multi-agent system for policy-based configuration management.

Cfengine ties observation or monitoring of a system to corrective operations, in a single repetitive control loop. To carry out its manifesto, in the environment of an operating system, it has to mix discrete and continuum approximations to system state into a unified model. In this paper, the cfengine agent is described from the viewpoint of control regulation.

A. Key ideas in this text

• *Policy* (P) is a description of intended host configuration. It comprises a partially ordered list of operations or actions. Cfengine policy is that part of system policy that can be coded into the host itself.
• *Configuration* (C) is the current state of the objects described by policy.
• *Operators* ($\hat{O}$) or primitive *actions* are the commands that carry out maintenance checks and repairs. They are the basic sentences of a cfengine program. They describe *what* is to be constrained and *how*.
• *Classes* are a way of quantifying the complex environment into discrete ('digital') regions that can be referred to by a symbolic identifier. They are constraints on the degrees of freedom available in the system parameter space. They are an integral part of specifying rules. They describe *where* something is to be constrained.
• A cfengine *state* is a point within the total system parameter space. States have the form:
`(object,attribute,value)`
We shall elaborate on these ideas below.

B. Cfengine and regulation philosophy

Cfengine assumes that changes of state occur unpredictably at any time, due to external 'disturbances'. It must therefore execute a 'continual' loop of observation and maintenance in order to achieve its policy state.

In control theory, systems are thought of as being continually regulated in order to optimize goals, by regulating certain state variables, in a cycle directly analogous to the one mentioned above. The key difference between this and cfengine is that cfengine employs two related strategies for implementing its goals.

• Deterministic, policy based regulation state.
• Stochastic, policy based regulation of state using average behaviour.

Both of these strategies can be considered as types of 'controllers'.

Cfengine holds to a set of principles, referred to as the *immunity model*[2], for seeking correctness of configuration. These embody the following features:
• Centralized policy-based specification, using an operating system independent language, which conceals implementation details.
• Distributed agent-based action, in which every host node is responsible for its own maintenance.
• Convergent semantics encourage every transaction to bring the system closer to an 'ideal' average-state, like a ball rolling into a potential well (negative feedback).
• Once the system has converged, action by the agent desists, or more usually, does not even start at all, when convergence was assured on a previous run of the agent.

The last two points are the most important. Most configuration agents either require a human to initiate change or rewrite the same constant configuration many times. In an analogous way to the healing of a body from sickness, cfengine's configuration approach is to always move the system closer to a 'healthy' state[3], or oppose unhealthy change: hence the name 'immunity model'. This idea shares several features with to the security model proposed in refs. [4], [5], and with control theory[6].

A 'healthy state' is defined by reference to a local policy. When a system complies with policy, it is healthy; when it deviates, it is sick. Cfengine makes this process of discrete 'maintenance' into an error-correction channel for messages belonging to a fuzzy alphabet[7], where error-correction is meant in the sense of Shannon[8].

C. *Classes and environment*

Cfengine uses the idea of classification to characterize a distributed environment into overlapping sets for time and location.

A class based decision structure is possible because a cfengine agent observes or *senses* the environment on every host on the network individually. Each host knows its own name, the type of operating system it is running and can determine whether it belongs to certain groups or not. Each host which runs a cfengine agent therefore builds up a list of its own discretized attributes (called the classes to which the host belongs). Classes that are meaningful in the context of a particular host include:

1. The date, time or identity of a machine, including hostname, address, network, and operating system and architecture of the host. This is for localization control.
2. An abstract user-defined group to which the host belongs for abstraction.
3. The result of any proposition about the system state.
4. Digitized observations of average performance, learned over time.
5. The logical combination of any of the above, with AND (`.`), OR (`|`), NOT (`!`) and parentheses.

One can think of this as a projection of the environment onto a discrete set of abstract classifiers. The classifiers form a patchwork covering of the environment and each class becomes a descriminating decision for controllers.

II. Policy and its interpretation

In an appropriate sense, policy is a set of grammatically structured control knobs for altering the average state of a system. The view of policy taken in ref. [9] is that of a series of instructions, coded into the computer itself, that summarizes the *expected* behaviour. The precise behaviour is not enforcable, since the system is not deterministic: it is subject to a number of environmental disturbances.

Policy is a really description about what we wish to be 'normal' about a system. A description of normality is a decision about how we define anomalies. There is thus an obviously link with anomaly detection[10]. If we cannot equate normality with policy then we have not even a partially predictable system to manage and the concept of 'management' would be meaningless.

This is where the split between system and environment has a fundamental conceptual bearing on our description of it. There are two kinds of normality that pertain to:

• Robust properties that we feel confident in deciding for ourselves (permissions of files, processes etc). These are decided and enforced. Deviations from these 'digital' specifications can be repaired or warned about directly by Shannon-like error correction.

• Stochastic properties that are determined by the environment and must hence be learned (number of users logged in, the level of web requests). These have fluctuating values but might develop stable averages over time. These cannot normally be 'corrected' but they can be regulated over time (again this agrees with the maintenance theorem's view of average specification over time[9]).

Cfengine deals with these two different realms differently: the former by direct language specification and the latter by machine learning and by classifying (digitizing) the arrival process.

III. Feedback and error regulation loops

The Shannon communication model of the noisy channel has been used to provide a simple picture of the maintenance process[7]. Essentially, maintenance is the implementation of corrective actions, i.e. the analogue of error correction in the Shannon picture. Maintenance is rather more complex than Shannon error correction, however, since it is not immediately clear that there is a simple digital picture of information for a system policy.

What makes the analogy valid is that Shannon's conclusions are independent of a theory of observation and measurement that becomes essential for policy. For simple alphabetic strings, the task of observation and correction is trivial. However, the conclusions apply even for more complicated models of observation (monitoring) and correction because the conclusions do not depend on the nature of these actions.

A necessary and sufficient characterization of digital policy is provided by the computer science idea of a language[11]. A language is just a structured pattern of state information.

Defining policy in language theoretical terms allows one to model it as a stream of operational messages. All we need to do this is to create a one-to-one mapping between the basic operations of cfengine and a discrete symbol alphabet. e.g.

```
A -> ``file mode=0644''

B -> ``file mode=0645''

C -> ``start apache httpd''
```

The agent interprets and translates the policy symbols into actions through operations, also in one to one correspondence.

1. Cfagent observes : X
2. Policy says :$X \rightarrow A$
3. Agent invokes :$A \rightarrow \hat{O}_{\rm file}(\texttt{passwd}, \texttt{0644}, \texttt{root})$

Let us suppose that example above evaluates to the alphabetic symbol 'A'. When the agent observes these properties of the named object it comes up with a symbol value based upon what it has measured. Suppose now that a user of the system (who is formally part of the environment) accidentally changes the permissions of the password file from mode=0644 to mode=0600. Moreover, we can suppose that this new value evaluates to the alphabetic character 'X'. The corrective controller sends the message $X \to A$.

$$ABCXBC... \to ABCABC... \tag{1}$$

The message transmission medium in this process is time itself. We regard the system (as is normal in the physical sciences) as being propagated from its current location to exactly the same place, or possibly across a system bus. In other words, the time development of the system is just the transmission of the system into the future over no distance (see fig 1).

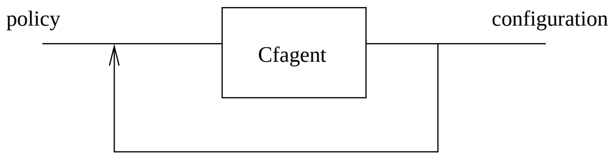

Fig. 1. Cfengine policy regulation as a controller. The state of the system configuration with respect to policy is being regulated given the disturbances created by users who make accidental or deliberate changes manually which do not fall within the limits set by policy.

The alphabet, as described, might appear infinite on casual reflection, in the sense that data objects that parameterize operations tie basic operations to a relativistic scheme of change – not to a fixed system. However cfengine is used in a single real system where all policies are actually finite. In other words, although the space of all possible policies is potentially very very large (though never truly infininte due to finite memory etc), only a small fraction of the possibilities is ever realized on a real system and this problem is not a limitation.

IV. Stochastic and deterministic scheduling

Many policy based systems use the Event-Condition-Action (ECA) model to regulate system state[13]. This is like a 'just in time' inventory model[14]. An alternative to this is to use a batch process of deterministic inventory scheduling, in which one plans maintenance checks at regular intervals, processing the backlog of events that have arrived (see fig. 2). The latter approach has greater time-uncertainty than an immediate triggered response, but it can perform all the same functions as the ECA model.

Cfengine uses both strategies, in order to cope with both determinism and non-determinism in its change processes. The probability of change in discrete system state is small, so a batch processing schedule has low uncertainty. The dynamical state of the system has a high rate of change, on the other hand, so a combination of event processing and batch processing is a more rational solution.

The aim of any controller's scheduling method is to limit the uncertainty in system state to a predictable interval of time. If state cannot be corrected in time, it is possible that the state will run away monotonically.

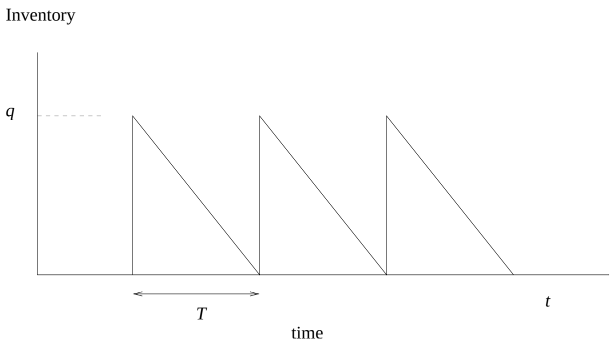

Fig. 2. A basic inventory model viewpoint with continuous cyclic operation shows how the maintenance 'stock' is realized 'suddenly' and depleted at a constant rate until it is exhausted.

V. Configuration management and convergence

Stability and convergence of discrete alphabetic states requires there to be 'absorbing states', i.e. for operations to behave like semi-groups[15], [12]. This is the analogue of stable negative feedback.

Another way of implementing this is through convergent operations. Cfengine uses the idea of *convergence* to an ideal state. This means that, no matter how many times cfengine is run, its state will only get closer to the ideal configuration. This is a stronger condition than *idempotence* as in Couch's interpretation[15], [16].

Note that the point of convergence is that multiple orthogonal, convergent operations will always lead to the correct configuration, no matter which part of the configuration is incorrect, or in what order things occur. Complex operations might not complete within a single scheduled iteration, if external factors intervene in an untimely manner; but they will always converge eventually. This is proven in ref. [2].

Property 1: Regardless of how long it takes, or of the scheduling order, a configuration will be implemented without requiring procedural logic. Operations are thus self-ordering, as long as all of the operations are convergent and orthogonal. This provides a notion of atomicity, and transactional security.

If two control operations are *orthogonal*, it means that they can be applied independently of order, without affecting the final state of the system. Using a linear representation of vectors and matrix valued operators, this is equivalent to requiring their commutativity.

VI. Anomaly Research

There is no system available in the world today which can claim to fully detect and classify the functioning state of a computer system. Cfengine does not attempt to provide a "product" solution to this problem; rather it incorporates a framework, based on the current state of knowledge, for continuing research into this issue.

In cfengine, an extra daemon (cfenvd) is used to collect statistical data about the recent history of each host (approximately the past two months), and classify it in a way that can be utilized by the cfengine agent. Data are are gradually aged so that older values count less[17], [10]. In this way, cfengine can learn over time the normal state of a computer system in a dynamical environment. Observations can be made of any measureable scalar value.

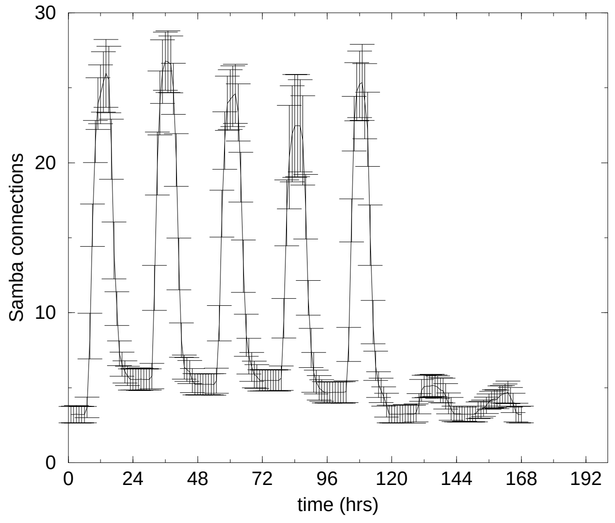

Fig. 3. Data learned about past behaviour (with standard deviation error bars) of incoming web request events as learned by cfenvd. We see a periodic pattern with peaks for Monday, Tuesday, etc, i.e. for each day of the week.

Cfengine has two definitions of abnormal or anomalous: an intermediate definition based entirley on previous observation, and a policy defined definition which is an expression of user wishes. Initially, cfengine calculates a smart 'running average' of every observable[10] and compares it to a geometric series of previous observations. It can then compute a measure of the variance of the data. The square root of the variance provides a base length-scale by which to classify the current value relative to the current average. Thus cfengine decides whether a current value is, for instance, more than two standard deviations above average for the current time of day. It is then up to the user to determine using a policy expression (see below) whether the actual deviation is to be considered sufficiently anomalous or not to warrant a response.

The current definition of normal is automatically adapted to the changing conditions, using a form of unsupervised learning, which has a built-in inertia to prevent anomalous signals from being given too much credence. Persistent changes will gradually change the 'normal state' of the host over an interval of a few weeks. Unlike some systems, cfengine's training period never ends. It regards normal behaviour as a relative concept, which has more to do with local stability than global constancy.

Cfenvd links the average behavioural world with the discrete control-knob world by setting classes which describe a digitized view of the current average state of the host in relation to its recent history. The classes describe whether a parameter is above or below its average value, and how far from the average the current value is, in units of the standard-deviation[10]. This information could be utilized to arrange for particularly resource-intensive maintenance to be delayed until the expected activity was low.

For instance, a practical policy for data copying could be to avoid times at which server load is especially high. Since significant server activity could place a difficult load on a host one could explicitly avoid times of high activity.

```
copy:

!www_in_high_dev3:: # def anomaly

  /www-user-database  dest=/www-backup
```

The example here suggests that this rule should not be executed if incoming World Wide Web activity is three standard deviations or more above the norm. Measures like the standard deviation are rather one dimensional however. We might rather be interested in using the shape of a histogram of sample data as a control switch. For example, consider the two (sideways) histograms of IP addresses extracted from a network data sample. The significances of the sharp left distribution or the blunt right distribution are quite different. In an the case of high load, the sharp (low entropy) distrbution could be viewed as an attack, while thigh high entropy right distribution simply unfortunate. The Shannon entropy of the distribution of originating IP

```
IP 1:   *                    ***
IP 2:   **                   **
IP 3:   **                   ***
IP 4:   *****************    ***
IP 5:   **                   *****
IP 6:   *                    **
IP 7:   *                    ***
IP 8:   *                    **
IP 9:   *                    ***
```

addresses has been used to predicate anomalies based on sample statistics[10]:

```
 icmp_in_high_anomaly
       & !entropy_icmp_in_low::

   ShowState(incoming.icmp)
```

One way of understanding cfengine's behaviour in relation to environment and policy is as a Markov fluctuation model of change[18], seeking an equilibrium configuration.

The advantage of Markov process regulation is that it requires no controller memory (or perhaps just a short memory). On the other hand, not all processes can be understood in this way. A controller which is to handle inhomogeneous patterns of change must be able to record at least some information about the past pattern of change.

Cfengine uses a small memory, short history Markov approximation, so that is can use what is known about weekly periodic system patterns[19] to optimize[10]. It maintains only a 'shadow of the past', not a detailed time-series record

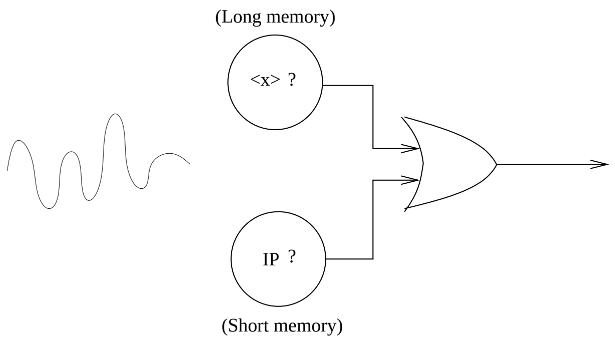

Fig. 4. A strategy of co-stimulation is used to sequentially filter information. First, long term (low grade) memory decides whether an event seems statistically significant and assess the likelihood of danger. If significant, short term (high grade) memory is used to recognize the source of the anomaly.

(see ref. [10] for details). Use of iterative updating allows one to build up a repository of long term and short term knowledge of the state of the system, with a minimum memory budget. The two can then be used in concert to classify system events i relation to policy. In other words, one can use qualifiers based on long and short term memory (see fig. 4) to decide when to respond to the classified anomaly. We call this co-stimulation.

The anomaly measurement scale is the standard deviation, based on collected data around the learned trend; but a single standard deviation is sometimes not even resolvable on a lightly used host, i.e. it could be less than the discrete counting scale of the events (half a process, ofr instance); the appearance of a single new event might trigger a standard deviation from the norm. This indicates the necessity of having a policy specification for what abnormal means in each case. On more heavily loaded hosts, with persistent loading, more reliable measures of normality can be obtained.

How can one avoid a deluge of 'false positives' in anomaly detection? We must invoke an anomaly policy to further classify events as interesting or uninteresting, using the information content of the events. As part of policy, we can combine the symbolic and numerical classifications of events with co-stimulation once again.

VII. Conclusion

Cfengine is a reactive agent which behaves like a discrete classical controller in a number of ways. It measures or samples a time-series at its input and is able to affect the output correspondingly, with parametric controls set by a policy specification. Cfengine is specifically geared towards an information theoretic interpretation of 'negative feedback', based on languages of the Chomsky hierarchy. The current cfengine works on data in scalar states and regular languages. Future versions will be able to understand context free languages also.

References

[1] M. Burgess, "Evaluation of cfengine's immunity model of system maintenance," *Proceedings of the 2nd international system administration and networking conference (SANE2000)*, 2000.

[2] M. Burgess, "Cfengine's immunity model of evolving configuration management," *Science of Computer Programming*, vol. 51, pp. 197, 2004.

[3] M. Burgess, "Computer immunology," *Proceedings of the Twelth Systems Administration Conference (LISA XII) (USENIX Association: Berkeley, CA)*, p. 283, 1998.

[4] P.D'haeseleer, S. Forrest, and P. Helman., "An immunological approach to change detection: algorithms, analysis, and implications," *In Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy (1996)*.

[5] A. Somayaji, S. Hofmeyr, and S. Forrest., "Principles of a computer immune system," *New Security Paradigms Workshop, ACM*, vol. September 1997, pp. 75–82.

[6] J.L. Hellerstein, Y. Diao, S. Parekh, and D.M. Tilbury, *Feedback Control of Computing Systems*, IEEE Press/Wiley Interscience, 2004.

[7] M. Burgess, "System administration as communication over a noisy channel," *Proceedings of the 3nd international system administration and networking conference (SANE2002)*, p. 36, 2002.

[8] C.E. Shannon and W. Weaver, *The mathematical theory of communication*, University of Illinois Press, Urbana, 1949.

[9] M. Burgess, "On the theory of system administration," *Science of Computer Programming*, vol. 49, pp. 1, 2003.

[10] M. Burgess, "Probabilistic anomaly detection in distributed computer networks," *Science of Computer Programming*, p. (To appear), 2005.

[11] H. Lewis and C. Papadimitriou, *Elements of the Theory of Computation, Second edition*, Prentice Hall, New York, 1997.

[12] M. Burgess, *Analytical Network and System Administration — Managing Human-Computer Systems*, J. Wiley & Sons, Chichester, 2004.

[13] N. Damiannnou, A.K. Bandara, M. Sloman, and E.C. Lupu, *Handbook of Network and System Administration*, chapter A Survey of Policy Specification Approaches, Elsevier, 2007 (to appear).

[14] H.L. Lee and S. Nahmias, *Logistics of Production and Inventory*, vol. 4 of *Handbooks in Operations Research and Management Science*, chapter Single Product, Single Location Models, Elsevier, 1993.

[15] A. Couch and Y. Sun, "On the algebraic structure of convergence," *LNCS, Proc. 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, Heidelberg, Germany*, pp. 28–40, 2003.

[16] A. Couch and Y. Sun, "On observed reproducibility in network configuration management," *Science of Computer Programming*, vol. 53, pp. 215–253, 2004.

[17] M. Burgess, "Two dimensional time-series for anomaly detection and regulation in adaptive systems," *IFIP/IEEE 13th International Workshop on Distributed Systems: Operations and Management (DSOM 2002)*, vol. LNCS 2506, pp. 169, 2002.

[18] G.R. Grimmett and D.R. Stirzaker, *Probability and random processes (3rd edition)*, Oxford scientific publications, Oxford, 2001.

[19] M. Burgess, H. Haugerud, T. Reitan, and S. Straumsnes, "Measuring host normality," *ACM Transactions on Computing Systems*, vol. 20, pp. 125–160, 2001.