**QUEEN'S
UNIVERSITY
BELFAST**

Introducing Automated Management Through Iteratively Increased Automation and Indicators

McLarnon, B., Robinson, P., Milligan, P., & Sage, P. (2011). Introducing Automated Management Through Iteratively Increased Automation and Indicators. In *Proceedings from the IFIP/IEEE International Symposium on Integrated Network Management* (pp. 1116-1121) <https://doi.org/10.1109/INM.2011.5990522>

Published in:

Proceedings from the IFIP/IEEE International Symposium on Integrated Network Management

Queen's University Belfast - Research Portal:

[Link to publication record in Queen's University Belfast Research Portal](#)

General rights

Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy

The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

Introducing Automated Management Through Iteratively Increased Automation and Indicators

Barry McLarnon^{*†}, Philip Robinson^{*}, Peter Milligan[†] and Paul Sage[†]

^{*}SAP Research Belfast

Northern Ireland Science Park

Belfast, Northern Ireland

Email: barry.mclarnon@sap.com, philip.robinson@sap.com

[†]Queen's University Belfast

University Road

Belfast, Northern Ireland

Email: p.milligan@qub.ac.uk, p.sage@qub.ac.uk

Abstract—Introducing automation into a managed environment includes significant initial overhead and abstraction, creating a disconnect between the administrator and the system. In order to facilitate the transition to automated management, this paper proposes an approach whereby automation increases gradually, gathering data from the task deployment process. This stored data is analysed to determine the task outcome status and can then be used for comparison against future deployments of the same task and alerting the administrator to deviations from the expected outcome. Using a machine-learning approach, the automation tool can learn from the administrator's reaction to task failures and eventually react to faults autonomously.

Keywords—adaptability; artificial intelligence; automation; deployment; management; neural networks; reliability;

I. INTRODUCTION

Automated system management is becoming increasingly important for large-scale networks, particularly in the case of cloud computing environments, where the physical resources that are being managed can potentially be distributed across the world. System administration without the use of some form of automated system management (ASM) tool is now very difficult as the size and complexity of computer systems has expanded significantly [1].

ASM tools help manage this complexity by adding a structured process to the task of management, allowing the administrator to create a set of policies that are communicated to the systems, or *operating environment* to be controlled, making the necessary changes without requiring the administrator to be on-site, and in some cases without requiring them to be involved in the deployment process at all. Such tools are increasingly ubiquitous in large businesses and data centers. One example is the open source tool, Cfengine, which claims to be in use by over 5000 businesses worldwide, including large companies such as AMD, Cisco and Nokia [2].

However, a major disadvantage of current approaches is that many ASM tools introduce additional management

overhead or abstraction, providing tools that use specific proprietary languages or a range of components, removing the administrator from the details of how the underlying management tasks are performed. This can make it more difficult for them to track where tasks have failed, as they may be left only with a single line alert, a raw output log, or even unaware that a task has failed until other related faults occur. Where such tasks fail, there is the potential that other dependent components may also be affected with propagation, causing system-wide failure. As the impact of even a single hour of downtime can have a significant financial cost for a large company [3], it is important to ensure that such failures can be quickly traced, and where possible, avoided. By keeping the administrator in control of the details of how the task is deployed, and alerted when errors occur, the risk of failure propagation can be reduced. However, in order to reduce overall management complexity it is still necessary to provide a framework for automation of management tasks.

This paper addresses these issues by proposing a system through which the administrator can transition to automated management in stages. Previously manual deployment tasks can be individually added to the proposed ASM tool, and for each iteration of the task, the raw output is stored and analysed, providing a structured, expandable dataset called an *indicator*, and by comparing the indicators to the following task iterations, it is possible to identify expected patterns for success, and alert the administrator to encountered or new fault conditions. By adding a machine-learning layer using neural networks, the ASM can create fault mitigation strategies through alerting the administrator to deployment errors and analysing the approach taken to recover.

The remainder of this paper is composed as follows: Section II details the related work in this area; Section III details the process of increasing automation in stages and how gathered feedback data can be used to inform the administrator of potential deployment issues; Section

IV looks at the introduction of a neural network to enable learned responses to faults; and finally, Section V offers a conclusion on the approach and gives an overview of the future direction.

II. RELATED WORK

This section provides an overview of related research in the field of system administration and the categories of automated system management tools that have been researched and developed to address the complexities of management.

A. System administration

Until recently, it was still common practice in system administration to take a “hands-on” approach, where an administrator would spend significant time creating ad-hoc scripts for individual repeated tasks, with little structure provided for moving to any kind of automation system [4]. This may be due to two issues. The first is trust in automation, where concerns have been raised about the loss of control over the operating environment due to automated system management [5] and the reliability and dependability of the automation solution itself [1]. The second is the issue of the *irony of automation* [6]; namely that introducing automation does not necessarily reduce management difficulty, and can, in fact, create more complexity [7], making this a barrier to entry for many administrators.

Collaborative tools have been suggested as a way of tackling the increasing knowledge base that administrators now need to be effective in their work [8]. While this can be very useful in gaining solutions to specific problems, much of the work performed during the day to day running of an enterprise system consists of frequently repeated but time-consuming tasks [9].

B. Automated management

Automated system management tools make the task of deploying and maintaining large-scale operating environments more streamlined and systematic by creating automated, structured processes of common tasks, such as resource selection, deployment and configuration, thereby reducing the workload of the administrator. This reduces the risk of faults introduced through human error, and allows the administrators to focus on other tasks such as policy management, with the benefit of reducing total cost of ownership (TCO), due to the lower maintenance and downtime costs associated with increased automation.

System management can be implemented with different degrees of automation, ranging from completely manual management to fully automated, with no input required from the administrator [10]. Researchers at IBM have identified five broad categories describing distinct levels of automation [11], which are widely accepted within the research community:

- 1) *Manual*: At this level, all system management and deployment tasks are performed by the administrator.
- 2) *Managed*: Some of the task steps are automated, with tools being used to provide the administrator with some of the input data that they need.
- 3) *Predictive*: The system is able to provide the required data and is able to recommend actions to the administrator.
- 4) *Adaptive*: Administrator interaction is reduced as the system monitors and controls itself, but alerts the administrator to scenarios that is unable to handle directly.
- 5) *Autonomic*: The system is able to manage and heal itself, with the administrator providing policy definitions, but being removed from the underlying system management.

As the degree of automation in system management increases, the complexity involved in creating such systems also increases, as they require more input data and knowledge of potential scenarios to reduce the need for administrator intervention.

Managed and predictive scripting environments such as LCFG [12] and Quattor [13] provide an environment for the manual creation of new or stored scripts, which can be applied to as many or as few nodes in the operating environment as required, with a centralised configuration file that describes the entire operating environment. The benefit of this approach comes in having a way to monitor and deploy the scripts that have been created across the network. By including monitoring agents in each node, the administrator can be alerted to their current status, which can help them make decisions in how to control the environment. However, this is still a highly involved process for the administrator, as they must use meticulous planning to ensure that the scripts perform the operations as expected. This leaves the ASM open to the influence of human error, such as deploying scripts to the wrong node or incorrect configuration.

Adaptive ASM solutions use rule-based management to allow the administrator to provide global and specific rules for the deployment and configuration of nodes in their operating environment. Tools such as Cfengine [14] and Puppet [15] consist of a centralised policy distribution server with a set of agents on each node that connect to the server and update their local policies according to their assigned definitions. These policies are written in a declarative language, allowing the administrator to describe a problem at a high level and let the agents complete the task of deployment and configuration without their intervention. The drawbacks of this approach are that the administrator is obscured from what is happening on each node, as they are individually responsible for ensuring their configuration is correct, and there is little feedback to the administrator of the current status of a management task, particularly when errors occur, increasing the recovery time.

The highest level of automation is provided by an autonomic system. Autonomic computing is an initiative associated most commonly with IBM [16][17]. It describes the ability of a system to self-configure, self-heal, self-optimize and self-protect. This initiative is used as the basis of the IBM Tivoli Management Framework [18]. Each managed element in the operating environment, whether hardware or software, is managed using a control loop, which is responsible for monitoring the element state and making decisions on how to manage it, based on a set of administrator-defined policies. However, it has been noted that self-management can be a problem in itself due to the increased complexity and the need for management of management [19]. Autonomic systems also remove the administrator from many of the regular tasks of management, which can mean that a fault in the autonomic system can be difficult to trace and errors caused by this can propagate through the environment as the administrator is finding the original problem.

III. ITERATIVE AUTOMATION WITH INDICATORS

As outlined in Section II, performing manual system management is difficult due to the size and distribution of modern computing infrastructures. Conversely, while moving to an ASM tool that offers a higher level of automation provides a way to more effectively manage the complexity of large-scale operating environments, it has the side-effect that it abstracts the underlying deployment task details from the administrator, which could potentially make faults harder to trace. By taking the novel approach of increasing automation over time, it is possible to keep the administrator in control of the automated tasks, while gradually reducing their involvement in the automatable decisions and unimportant output data.

Iterative automation can be described as the process of increasing the level of automation over time, starting with the administrator performing the manual steps of a management task in a managed ASM tool, and then increasing automation in stages, the objective of which is to reduce the occurrence of routine management tasks and redundant, overloaded messages, while enabling automated recovery strategies learned from the administrator's best practices. In order to avoid abstracting the administrator from the environment they are controlling, the manual instructions should be preserved as a parameterised script, allowing them to perform the same commands repeatedly. By adding a predictive layer, including monitored data from the operating environment and a set of rules, the ASM can then make decisions on how to configure the task and what nodes to target, further removing the administrator, but keeping the same set of instructions.

However, as most management tasks complete successfully once the steps have been defined correctly and most of the output is repeated frequently, the administrator no longer needs to analyse the output of every step. By allowing the

administrator to configure how much of the task output they receive, they are better able to prioritise their time to cope with errors. This should be possible on a task-by-task basis, as, for example, the deployment of one piece of software may have a completely unique set of issues, even with a different version or configuration of the same software. While the administrator may only need to be alerted if an error arises, it can be difficult to know which output pertains to the error. In the example of software deployment, the syntax for the logging of errors for one piece of software might be completely different to another. By creating a process where error output can be differentiated from expected output, varied task outputs can be analysed consistently.

As part of the process of increasing automation, a set of data associated with each deployed task is aggregated and improved over time. In the proposed system, this is described as an *asset*, which is stored in a database and can be used by the ASM tool to deploy a task multiple times. An asset consists of a number of connected pieces of information:

- *Descriptor*: Provides an overall description of an asset, such as its name and type, along with any additional parameters that are unique to the asset, for example, specific configuration options.
- *Resource*: Points to where any resources that are needed for the deployment of the asset are stored. This can be a package of software binaries, a virtual machine disk image or operating system install disk.
- *Command*: Describes the commands that can be performed with the asset, such as installation, removal or reconfiguration. This is linked to a script template which can be used to perform the associated command when deployed as a management task.
- *Indicators*: For each command, there are an associated set of indicators which describe the expected output, providing feedback and alerting the administrator to deviations from this output.

Indicators are built using the output data from the process of deploying the asset. The raw output is parsed to create a specific structure, allowing very different task output sets to be stored and referenced in a generic manner. Each indicator has the following structure:

- *ID*: A unique identifier for referencing each indicator.
- *Output Type*: Class of message that is output, thus alerting the administrator to the status of the deployment. There are five general classes that all messages are stored as: Information, Success, Warning, Error and Unknown.
- *Action Type*: The type of action that the output data describes. This is based on a set of administrator-defined keywords that are used during parsing to identify what instruction is being performed. If there is no discernible action type, the indicator defaults to 'none'.
- *Condition*: The set of conditions under which the indicator describes the current output.

Figure 1. Output data from multiple runs of a given task combined to form Indicators

- Occurs - A numbered range of the frequency a parameterised message may optionally occur in sequence.
- Includes - If the output includes a particular indicator ID, then this indicator will also occur.
- Fail - If the command described by a particular indicator ID has failed, this indicator will describe the fault output associated with it.
- *Message*: A parameterised form of the expected message which is used for comparison with the current output.

As the same task is performed multiple times, the indicators are expanded with any changes in the output data, such as previously unseen error conditions, as shown in Figure 1. This analysis involves identifying which lines of output indicate a successful deployment step and which provide a warning of problems that have occurred, along with their severity. Variables can also be introduced where the output will vary between runs, such as timestamps or file paths.

Using a system of Deployment With Indicators (DWI), the set of indicators can then be used in the action deployment process, where they are compared to the raw output data from the deployed task, both improving the indicators and feeding any relevant information about the status of the task back to the administrator. The process is shown in Figure 2 and the stages are outlined below:

- 1) The administrator defines an asset, providing a script template to perform the deployment action and creating a link to any required resources. Assets may be modified and refined as automation increases.
- 2) The administrator requests the deployment of an asset through the ASM tool, adding any additional configuration parameters and targets as required.
- 3) The ASM tool retrieves the asset data required for the deployment and fills in the parameters in the script template.
- 4) The finalised script is sent to the deployment daemon on the target system and executed, fulfilling the de-

Figure 2. Using indicators in the deployment process for improved feedback and increasing automation over time

ployment request.

- 5) The daemon responsible for running the deployed script gathers the raw output data and returns it to the analysis component.
 - 6) The analysis component retrieves the indicators associated with the deployed asset and compares the raw output line-by-line to them, checking for matches and inconsistencies. Any previously-unseen output is parsed for parameter variations and error conditions.
 - 7) The indicators are updated with any new output and condition changes.
 - 8) A summary of the output is returned to the automation tool, highlighting warnings, error conditions and unexpected data.
 - 9) The administrator receives some, most or all of this output, depending on the level of task feedback that they configure (on a per asset, deployment task, or global level).
- As the administrator is deploying management tasks, a neural network structure is operating with the data from the output analysis component. Using machine-learning techniques, it acquires the knowledge on how to tolerate and recover from task deployment errors, eventually without direction from the administrator:
- 10) The neural network is provided with the output from the task as well as the commands that created the output.
 - 11) Initially, the administrator is alerted to unlearned scenarios and is requested to input classification for the errors that were encountered and a new deployment task to mitigate the faults of the initial task. This process is repeated until a learning threshold has been met.
 - 12) Once the threshold has been met, the neural network is able to interface directly with the ASM tool and create its own fault mitigating management tasks based on the learned responses to failure scenarios.

IV. MACHINE-LEARNING FOR ITERATIVE AUTOMATION

Neural networks consist of simple processing elements that can accomplish complex computations by interconnecting with each other. The earliest work focused on

engineering and investigating Neural Networks is attributed to McCulloch and Pitt in their 1943 publication of a logical calculus for nervous activity [20]. One application of neural networks is in solving classification problems. A classification problem concerns how to identify patterns in raw input data elements $X = (x_1, \dots, x_n)$ and assign them to higher level, a priori labels $Y = (y_1, \dots, y_n)$. It is then easier to make decisions using the higher-level labels as opposed to the raw input data. The goal is to discover functions F that map the data to the labels, $F : X \rightarrow Y$.

To apply this towards realising iterative automation, a single layer, supervised neural network was designed, which uses basic boolean logic for activation rules and a feed-forward algorithm for learning and automation. There are three neurons in this layer, each of which take a pre-correlated command and output from a deployed task as the two inputs. These three neurons are defined below with their different activation rules, memory updates and outputs, the outcomes of which are summarised in Table I.

- **Help-Required (H):** Neuron sends a request to the human administrator for help if a pattern of command and output arrives that has failed and there is no mitigation in memory.
 - Activation Rule:
 - * IF Pattern(Command, Execution-Output) is new OR Result=? and Response=? THEN Fire Output (timestamp, command, execution-output)
 - Output: request for supervisory feedback from administrator
 - * At [timestamp] Command [command] resulted in condition [output]
 - * IF [Success?] THEN store Command + Output to memory ELSE get new mitigation command from administrator and store
- **Notification (N):** Neuron sends notifications to the administrator if there are known patterns but only sufficient knowledge to state what the result is (for example, a fault condition).
 - Activation Rule
 - * IF Pattern(Command, Execution-Output) is known and Result=success—fail and Response=? THEN Fire Output (timestamp, command, execution-output)
 - Output: request for more supervisory knowledge from administrator e.g. switch feedback off or, in case of fail conditions, introduce a mitigating response
 - * At [timestamp] Command [command] resulted in condition [output]
 - * IF [Success?] THEN store Command + Output to memory ELSE get new mitigation command from administrator and store
- **Automated Action (A):** Neuron sends new requests to the automated manager, as the pattern is known and

Table I
LEARNED NEURON RESPONSE PATTERNS FROM ACTIVATION RULES

Pattern		Knowledge		Neurons		
Command	Execution Output	Result	Response	H	N	A
x1	(timeout)	?	?	1	0	0
x1	y1	?	?	1	0	0
x1	y1	fail	?	0	1	0
x1	y1	fail	x2	0	0	1
x2	y2	?	?	1	0	0
x1	y2	success	1	0	1	0
x1	y2	success	0	0	0	0

there is sufficient knowledge for deciding what to do with success or fail conditions.

- Activation Rule
 - * IF Pattern(Command, Execution-Output) Result=fail and Response=automateRecovery THEN Fire Output (timestamp, command, execution-output)
- Output: send learned fault mitigation response to automated manager to be deployed
 - * At [timestamp] Command [command] resulted in condition [output]
 - * IF [Success?] THEN log Output ELSE create Command [command2]

By feeding this neural network with the commands and output from all tasks deployed using the ASM tool and letting the administrator provide feedback and alternative commands in situations where the suitable response cannot be extrapolated from the existing knowledge base, it is possible to ensure that, given the same conditions in a future deployment in a task, the ASM tool will respond to faults in a predictable, pre-determined manner without requiring further intervention by the administrator.

V. CONCLUSION AND FUTURE WORK

At this point in the research, a custom automated system management tool has been built that follows the principles outlined in this paper, including the process of iterative automation through the definition of assets and using these as a method of performing previously manual tasks, allowing output data to be captured and analysed. Indicators have been created from this data and used for comparison against the output of further deployments of each task.

This approach for output data analysis is novel, and when compared to other existing automated system management tools, improves the quality of feedback by providing the administrator with a way of controlling how involved they are in the deployment process. They do not necessarily have to be involved in the automated task directly, and can instead choose to be provided with an alert.

By including a neural network to gather knowledge from the management task output and allowing it to learn from the responses of the administrator, it has been shown that

the level of automation can be increased further, but allowing the administrator to remain in control of how the automated system management tool will respond to different fault conditions. To evaluate this approach, it is planned to compare the level of automation against the quality of feedback received for the custom ASM tool and analyse this in relation to a range of other existing ASM approaches.

In future, the neural network can be expanded to use backwards propagation in order to perform analysis of the root cause of faults that occur during the deployment process. The goal of this would be suggest possible mitigation strategies to the administrator, and eventually, to allow the ASM tool to respond to some previously unseen faults without the involvement of the administrator.

ACKNOWLEDGEMENT

The authors would like to thank the European Union-funded GEYSERS Project (FP7 INFOS-ICT-248657) for their contribution and feedback on this work.

REFERENCES

- [1] E. A. Anderson, "Researching system administration," Ph.D. dissertation, Carnegie Mellon University, 2002. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.1815>
- [2] Cfengine: Companies. Accessed 7th December 2010. [Online]. Available: <http://www.cfengine.com/pages/companies>
- [3] R. Nolan and F. W. McFarlan, "Information technology and the board of directors," *Harvard Business Review*, vol. 83(10), pp. 96–106, 2005.
- [4] R. Barrett, E. Kandogan, P. P. Maglio, E. M. Haber, L. A. Takayama, and M. Prabaker, "Field studies of computer system administrators: analysis of system management tools and practices," in *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*. New York, NY, USA: ACM, 2004, pp. 388–395.
- [5] P. P. Duez, M. J. Zuliani, and G. A. Jamieson, "Trust by design: information requirements for appropriate trust in automation," in *CASCON '06: Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*. New York, NY, USA: ACM, 2006, p. 9.
- [6] L. Bainbridge, "Ironies of automation," *Automatica*, vol. 19, no. 6, pp. 775–779, November 1983. [Online]. Available: [http://dx.doi.org/10.1016/0005-1098\(83\)90046-8](http://dx.doi.org/10.1016/0005-1098(83)90046-8)
- [7] A. B. Brown and J. L. Hellerstein, "Reducing the cost of it operations: is automation always the answer?" in *HOTOS'05: Proceedings of the 10th conference on Hot Topics in Operating Systems*. Berkeley, CA, USA: USENIX Association, 2005, pp. 12–12.
- [8] E. M. Haber and J. Bailey, "Design guidelines for system administration tools developed through ethnographic field studies," in *CHIMIT '07: Proceedings of the 2007 symposium on Computer human interaction for the management of information technology*. New York, NY, USA: ACM, 2007, p. 1.
- [9] B. Dijkster, "A day in the life of system administrators." [Online]. Available: <http://www.sage.org/field/ditl.pdf>
- [10] T. Sheridan and R. Parasuraman, "Human-automation interaction," *Reviews of human factors and ergonomics*, vol. 1, no. 1, pp. 89–129, 2006.
- [11] IBM, "Ibm global services and autonomic computing," 2002. [Online]. Available: <ftp://ftp.software.ibm.com/software/tivoli/whitepapers/wp-igs-autonomic.pdf>
- [12] P. Anderson and A. Scobie, *LCFG: The Next Generation*, Division of Informatics, University of Edinburgh, 2002.
- [13] R. García Leiva, M. Barroso López, G. Cancio Meliá, B. Chardi Marco, L. Cons, P. Poznański, A. Washbrook, E. Ferro, and A. Holt, "Quattor: tools and techniques for the configuration, installation and management of large-scale grid computing fabrics," *Journal of Grid Computing*, vol. 2, no. 4, pp. 313–322, 2004.
- [14] M. Burgess, "A tiny overview of cfengine: Convergent maintenance agent," in *Proceedings of the 1st International Workshop on Multi-Agent and Robotic Systems, MARS/ICINCO 2005*, 2005.
- [15] Puppet documentation. Reductive Labs. Accessed 7th December 2010. [Online]. Available: <http://reductivelabs.com/trac/puppet>
- [16] IBM, "An architectural blueprint for autonomic computing, 3rd edition," 2005. [Online]. Available: <http://www-03.ibm.com/autonomic/pdfs/ACBlueprintWhitePaperV7.pdf>
- [17] M. C. Huebscher and J. A. Mccann, "A survey of autonomic computing—degrees, models, and applications," *ACM Comput. Surv.*, vol. 40, no. 3, pp. 1–28, August 2008.
- [18] G. Lanfranchi, P. Della Peruta, A. Perrone, and D. Calvanese, "Toward a new landscape of systems management in an autonomic computing environment," *IBM Systems Journal*, vol. 42, pp. 119–128, 2003.
- [19] K. Herrmann, G. Muhl, and K. Geihs, "Self management: the solution to complexity or just another problem?" *Distributed Systems Online, IEEE*, vol. 6, no. 1, 2005.
- [20] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.