Policy Generation Framework for Large-Scale Storage Infrastructures

Ramani Routray*, Rui Zhang*, David Eyers†, Douglas Willcocks‡, Peter Pietzuch‡, Prasenjit Sarkar*
*IBM Research – Almaden, USA {routrayr, ruizhang, psarkar}@us.ibm.com
†University of Cambridge, UK {first.surname}@cl.cam.ac.uk
‡Imperial College London, UK {dtw107, prp}@doc.ic.ac.uk

Abstract

Cloud computing is gaining acceptance among mainstream technology users. Storage cloud providers often employ Storage Area Networks (SANs) to provide elasticity, rapid adaptability to changing demands, and policy based automation. As storage capacity grows, the storage environment becomes heterogeneous, increasingly complex, harder to manage, and more expensive to operate.

This paper presents PGML (Policy Generation for large-scale storage infrastructure configuration using Machine Learning), an automated, supervised machine learning framework for generation of best practices for SAN configuration that can potentially reduce configuration errors by up to 70% in a data center. A best practice or policy is nothing but a technique, guideline or methodology that, through experience and research, has proven to lead reliably to a better storage configuration. Given a standards-based representation of SAN management information, PGML builds on the machine learning constructs of inductive logic programming (ILP) to create a transparent mapping of hierarchical, object-oriented management information into multi-dimensional predicate descriptions. Our initial evaluation of PGML shows that given an input of SAN problem reports, it is able to generate best practices by analyzing these reports. Our simulation results based on extrapolated real-world problem scenarios demonstrate that ILP is an appropriate choice as a machine learning technique for this problem.

I. Introduction

Forward thinking enterprises have widely accepted cloud computing. These enterprises gain agility in their ability to optimize their various IT drivers (storage, networking, compute, etc). Further, TCO (Total Cost of Ownership) is reduced, as the enterprises can operate on a "pay as you go" and "pay for what you use" basis. This has fueled the emergence of cloud providers such as Amazon S3 [1], EC2 [2], iTricity [3] and many others. Storage is an important component of the cloud and thus cloud providers often employ Storage Area Network (SANs) to offer scalability and flexibility. Management of all of the data center resources of a service provider is a complex task. In this paper, we focus on storage and specifically the SAN management aspect of it, although our solution can be generalized for other aspects of data center management.

Tools for managing data centers are built of basic building blocks, such as discovery, monitoring, configuration, and reporting. Advanced functions, such as replication, migration, planning, data placement, and so on, are built on top of those. Providing non-disrupted service to business-critical applications with strict Service Level Agreements (SLAs) is a challenging task. It requires careful planning, deployment, and avoiding single points of failure in configurations. Adherence to best practices is essential for successful operation of such complex setups.

In these scenarios, experts rely on experience as well as repositories of *best practice guidelines* to proactively and reactively prevent any configuration errors in a data center. Best practices, or rules of thumb, are observed while planning for a deployment to proactively avoid any misconfiguration. In this way, valid configurations are planned and deployed. But at times, valid deployment in an individual layer (server, storage and network) incrementally or in fragmented fashion may lead to misconfigurations from an end-to-end datapath perspective, and require urgent, reactive validation. The terms "best practice", "policy" and "rules of thumb" are used interchangeably throughout this paper, and are all treated as having the same meaning as each other.

For example, consider a database-driven OLTP application running on multiple servers with operating system A that access storage from a Storage Controller B through a fiber channel fabric. Initial provisioning and deployment were done by server, network, storage and database administrators. The same task might also have be done through automated workflows. During the application planing and deployment, best practices were followed by a network administrator to create zoning [4] configurations such as (i) *Devices of type X and devices of type Y are not allowed in the same zone*, and (ii) *No two different host types should exist in the same zone*. Later, due to the application requirements, a server administrator changed the operating system of one of the servers to A'. All the best practices for server and application compatibility were ensured by the server administrator. But the host bus adapters (HBAs) associated with the server still remained in its previous zone, violating both of the fabric policies stated above, and resulting in data loss.

IBM has a Storage Area Network (SAN) Central team whose job is to examine all known storage area network configuration issues and come up with best practices—see the appendix of [5]. These best practices have helped the SAN Central team to reduce the time required to resolve configuration errors from 2 weeks to 2 days as 80% of the configuration problems are caused by the violation of a few best practices. However, manual generation of the best practices is costly, requiring 20 man-years of data gathering and analysis so far. Sharing of best practices and collaboration across cloud providers to proactively prevent configuration errors—or at least to quickly react to problems—is also another important, related aspect of this process [6].

In this paper, we present an automated policy generation framework that uses Inductive Logic Programming (ILP) to learn and generate storage-specific best practices. Input to PGML is a relational system management repository and a set of problem reports raised against the same managed environment. Output of this system is a set of best practices with confidence values associated with them. These best practices are then validated by field experts. Internally, PGML creates an innovative mapping of relational information to ILP constructs guided by storage-specific domain knowledge.

Challenges: The challenges in generating best practices for a domain are numerous. There needs to be a systematic way to gather all the data from a complex and heterogeneous environment required for problem diagnosis. Success of any automated policy generation mechanism is determined by the quality and quantity of the data provided to it. Our data sets have a large number of entities, attributes and associations. This points to the need for dimensionality reduction so that the data sets can be analyzed efficiently. Many ML tools today can only deal with clean, well-formatted data. The cost of transforming raw data collected from management infrastructure into a form consumable by a ML tool should not be underestimated. In addition, such preprocessing is being done in an *ad hoc* manner for every new problem in every new data set. The time and effort spent to prepare the data largely outweighs the time spent to analyze it. In a way, while the state-of-the-art use of ML reduces the manual effort to analyze data, it is introducing significant man hours in "massaging" the data. Finally, the system would also want the purest subset of entities, attributes and associations that contribute to a configuration error. This would require the use of a highly accurate data classification tool that can overcome incomplete dimensions as well as noise contained within the data sets.

It is worthwhile noting that the problem of generating best practices is different from analyzing the root cause of a problem even though both techniques are valuable from an operational standpoint. In problem determination we are looking for the root cause specific to a single failure report, whereas in best practice generation, we find common root causes across multiple failure reports. A root cause analysis algorithm is a reactive mechanism designed to pinpoint the exact behavior of an entity or a set of entities that is causing a configuration problem. For example, if a set of computers have problems accessing a storage subsystem, a root cause analysis might reveal that computers with a certain operating system tend to overwrite the signatures of disks belonging to other computers. In contrast, best practice determination is a predictive mechanism that shows the minimal combination of entities and attributes that need to be followed so as to avoid a recurrence of a configuration problem. In the example above, the best practice would be to avoid putting computers with multiple operating systems in the same SAN zone.

Contributions: The contributions of this paper are:
(1) A novel framework that performs a transparent mapping of hierarchical, object-oriented system management information in CIM format into ILP-based, multi-dimensional predicate descriptions. PGML uses this layer to automatically generate best practices. The framework is unique in minimizing data preprocessing cost through the use of a machine learning technique that can naturally represent the multi-dimensional relationships inherent in storage systems management data
(2) The PGML Framework is based on an innovative workflow that streams information through layers without any data loss or creation of ambiguity while following guidance of the background knowledge. This layered workflow can easily be extended to apply to other domains of similar nature.
(3) An initial evaluation of this framework despite the limited availability of real world data. Also, the selection of ILP as the machine learning technique for this domain over others is justified.

II. Related Work

Failure diagnosis in distributed systems has been studied extensively [7]. Traditional approaches usually rely on explicit modeling of the underlying system. One widely used technique is the knowledge-based expert system. Such an approach may work well in a controlled, static environment. However, as the complexity of the system keeps growing, it becomes impossible to encompass all the necessary details. For instance, the device interoperability matrix is so dynamic that it is infeasible for an organization to build a comprehensive matrix at all. Applications on top of the network add another layer of complexity (e.g. disk and tape traffic cannot flow through the same HBA).

From the perspective of machine learning, failure diagnosis can be viewed as an anomaly detection problem [8], [9], [10], [11], particularly if the majority of the training samples are negative cases (i.e. non-problematic situations). The key idea is to describe what good data look like and define an anomaly to be a configuration data point that does not fit the profile of good data. The concept of feature selection

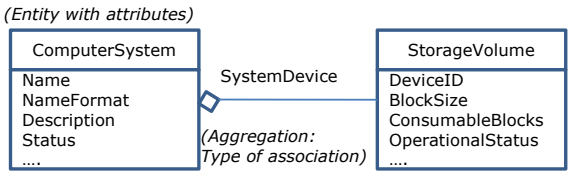

Figure 1. SMI-S Profile example

and model selection are also explored in several recent papers [12], [13]. Diagnosis of configuration problem has been studied in several areas such as the Windows Registry [14], [15], [9], [16], router configuration [17] and general Internet applications [18]. Other recent research activities focus on performance problems [19], [20], software failure [21], [22], general fault localization techniques [12], [23] and also automated policy generation for mobile networks [24].

III. Background

Exponential growth in storage in recent times have made storage management a monumental task for administrators. Popular storage/system administration offerings available in the marketplace, such as IBM TivoliStorage Productivity Center (TPC) [25], EMC Control Center [26], and HP System Insight Manager [27], address the task of seamlessly managing a complex and heterogeneous environment.

Enterprise storage setups require careful planning, deployment and maintenance during their lifetime. Guidance policies for proactive validation during planning, and validation policies for reactive validation later on are an important aspect of policy-based management. For example, Storage Planner [28] and Configuration Analyzer [29] components of IBM TPC are examples of the above paradigm. Usage of policies generated out of the observations made by field experts is a well known approach. Our framework augments the observation of human field experts by learning the patterns that are not easily visible given the scale of data to be observed.

The SMI-S (Storage Management Initiative Specification) [30] is an open, extensible, industry standard aimed at enabling the seamless management of multi-vendor heterogeneous equipments in a storage environment. SMI-S leverages the hierarchical and object-oriented architecture of CIM [31] and WBEM [32] to enable exchange of semantically rich management information of devices over the network. Most modern storage devices are SMI-S compliant and expose behavioral aspects of device management using standard profiles. Simple examples of the SMI-S concepts stating the entities, attributes and association are shown in Figure 1.

Management solutions discover the devices such as servers, network switches, storage controllers, tape libraries through the Service Location Protocol (SLP). Properties of the devices are then queried through a standard CIM client and are stored in a central relational repository; usually a database. This database contains correlated information across devices that gives a view of the end-to-end data path.

Our initial work in this area used decision trees for policy generation [33], and investigated sharing and collaboration of policy repositories across multiple data centers [6]—both have laid the foundation for creation of PGML.

IV. Overview of PGML

Let us consider a cloud provider environment that serves multiple customers, or an internal enterprise data center that serves multiple departments of a company. Each customer has its applications hosted on a set of virtual machines or servers, consuming storage from multiple storage controllers. The data flow will use multiple network switches along its data path. Based on customer workload requirements, deployments are planned and resources are allocated. Should anomalous behaviour be observed, the customer creates a problem ticket such as (i) *Application A is not accessible* or (ii) *Server A cannot write data on file system B*. PGML uses a commercial storage management server, IBM TivoliStorage Productivity Center [25], that collects monitoring data from multiple layers of the IT stack including databases, servers, and the SAN. Collected data are persisted as time-series records in a relational database. Figure 2 shows the building blocks of PGML along with the rest of the infrastructure stack on which it operates.

Servers have attributes such as *Name, IP Address, OS Type, OS Version, Vendor, Model,* and so on. Each Server has one or more Host Bus Adapters (HBAs). HBAs have attributes such as *WWN, Port Speed, Number of Ports, etc*. A Fiber Channel Fabric has one or more switches with attributes *WWN, Number of Ports,* and *Port to Port*. Storage Controllers have attributes like *Pools, Volumes, Extents, Disks,* and so on. Through *Port WWN, SCSI ID* of storage volumes and so on, end-to-end data path tuples are created. In basic terms, the structure of correlated end-to-end data path tuples are as follows: *(Server, HBA, Initiator Port, Fabric, Target Port, Storage Controller, Volume)*

In general, there are three kinds of attributes: (i) direct attributes, (ii) associations, and (iii) derived attributes based on domain knowledge. An example of the latter might be accessibility based on masking, mapping and zoning information.

In an operational environment, if a problem ticket gets created and it tags any element in the data path, the whole tuple is marked as a problematic one and is otherwise marked as a non-problematic one. Even though each device reports data individually and those data get stored in multiple database tables, the required data can be selected in one place, by utilising database views.

Having described the storage area network domain, we move to explain the rationale behind the selection of ILP as the machine learning technique. An approach using decision tree learning [33] was found not to be scalable for derived

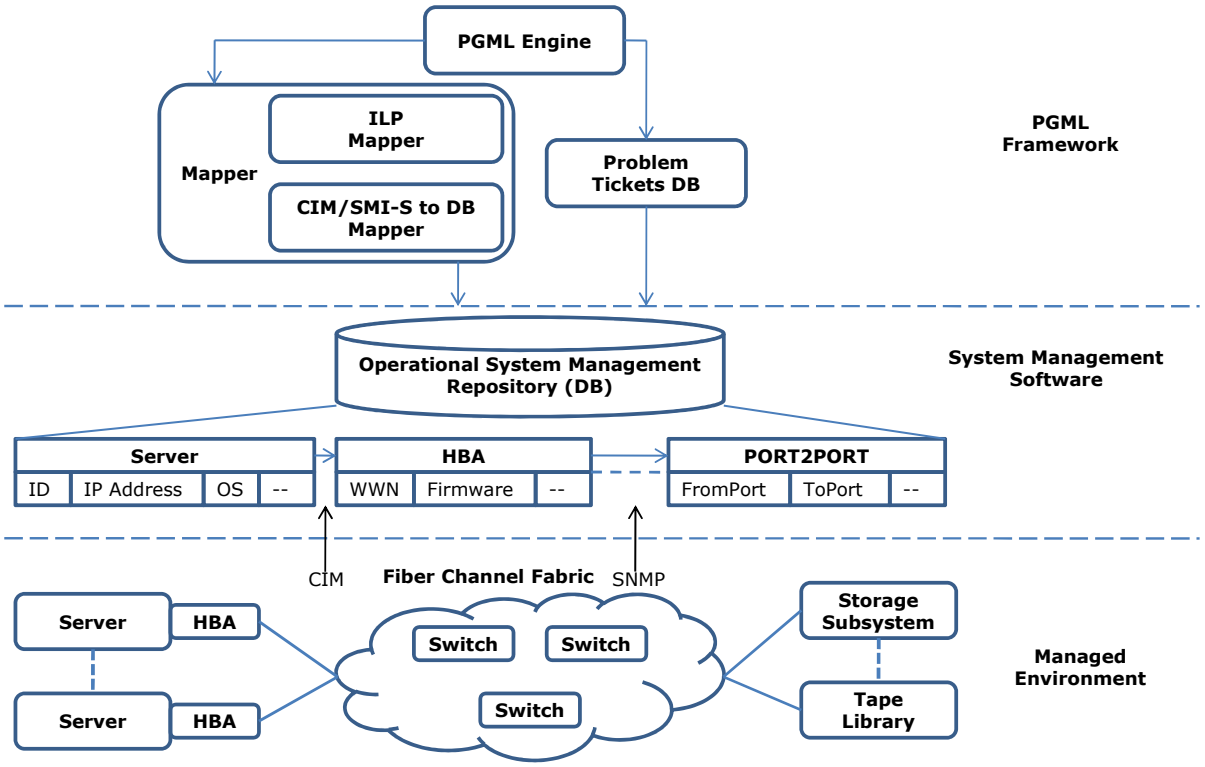

Figure 2. PGML System Overview

attribute creation. Further, preprocessing of the data for input in the decision tree learning software led to a semantic transformation that might contribute to potential data loss, and thus incorrect policy generation. For example, to learn a policy such as *Operating Systems A and B should not be in the same Zone*, we need to create derived attributes such as (i) a Boolean *heterogeneous* that is associated with each zone, (ii) attributes representing the presence or absence of a permutation of all of the combinations of two operating systems that might be in a zone. Creating such derived attributes for all *containment* associations such as zone is not very scalable. We have investigated use of multiple machine learning techniques and tools, namely Aleph [34], HR [35] and Progol [36]. Both Aleph [34] and Progol [36] are top-down relational ILP systems based on inverse entailment whereas HR [35] is a Java-based automated reasoning tool for theorem generation. Limitations, in terms of arithmetic support for comparison and cardinality in Progol or automatic generation of background knowledge in HR led us to use the Progolem ILP tool [37], [38], [39] for policy generation. ILP evaluates positive and negative examples based on background knowledge to generate *hypotheses*. Progolem generates the hypothesis in the form of first order logic expressions with quantifiers.

Next we describe the details of the input and output data and the components of our framework.

CIM/SMI-S to DB Mapper: Different data centers' cloud

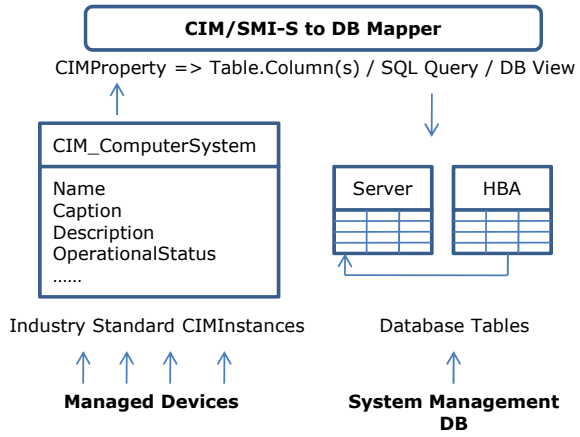

Figure 3. CIM/SMI-S to DB Mapping

administrators deploy a variety of system and storage management solutions to keep control of their environment. Each solution stores information regarding the managed environment in a relational database using a proprietary schema. Most of the time, the proprietary schema is influenced by the CIM/SMI-S schema but not all database schemas are the same. Since the underlying devices expose information according to an industry standard, we have defined a mapping layer that maps the management information in the database to the CIM/SMI-S information. This mapping is seamless, due to the native nature of the underlying models.

```
% Modes
:- modeb(*,fabric_has_switch(+fabric_id,-switch_id)).
:- modeb(1,switch(+switch_id,#codelevel,#model,-serialnumber,#type,#vendor)).
:- modeb(*,fabric_has_computer(+fabric_id,-computer_id)).
:- modeb(*,switch_has_fcport(+switch_id,-fcport_id)).
:- modeb(1,subsystem(+subsystem_id,#codelevel,#model,-serialnumber,#type,#vendor)).
:- modeb(*,hba_has_fcport(+hba_id,-fcport_id)).
:- modeb(1,hba(+hba_id,#drivername,-driverversion,#firmwareversion,#model,#vendor)).
:- modeb(1,computer(+computer_id,-hostname,#ostype,#osversion)).
:- modeb(1,fabric(+fabric_id,#exptype,-wwn)).
:- modeb(*,fcport_has_fcport(+fcport_id,-fcport_id)).
:- modeb(*,fabric_has_subsystem(+fabric_id,-subsystem_id)).
:- modeb(*,computer_has_hba(+computer_id,-hba_id)).
:- modeb(*,subsystem_has_fcport(+subsystem_id,-fcport_id)).
:- modeb(1,fcport(+fcport_id,-name,#portspeed)).
:- modeh(1,faultyfabric(+fabric_id)).

% Examples

%Positive Examples
example(faultyfabric(fabric_393),1).
example(faultyfabric(fabric_0),1).
.....

%Negative Examples
example(faultyfabric(fabric_458),-1).
example(faultyfabric(fabric_196),-1).
......

% Entities

% switch(switch_id,codelevel,model,serialnumber,type,vendor).
switch(switch_0,0,model_0,serialnumber_9,type_0,vendor_0).
....

% computer(computer_id,hostname,ostype,osversion).
computer(computer_0,hostname_0,ostype_3,3).
.....

% Relations

% fabric_has_switch
fabric_has_switch(fabric_327,switch_477).
...

% fcport_has_fcport connectivity
fcport_has_fcport(fcport_49694,fcport_49632).
...
```

Figure 4. Input format to the PGML Engine

This layer helps our tool remain agnostic to the multi-vendor management solutions. Some information regarding the managed environment is also collected through SNMP and is currently not captured by our mapper. An example of this mapping is shown in Figure 3.

ILP Mapper: This module retrieves the data from relational database based on the CIM/SMI-S mapper described above. Then, it transforms the CIM data into a format expected by the ILP engine. This module keeps the data preprocessing cost optimal through dimensionality reduction of given data sets. Entities, attributes, relationships, positive examples and negative examples are created through this module and are passed on to the ILP engine for hypothesis (and thus policy) generation. A sample of the input file format is shown in Figure 4.

PGML Engine: The PGML engine uses the Progolem tool for ILP. A formal definition of ILP states [39]:

Given background knowledge: B
Positive examples: E^+
Negative examples: E^-
Hypothesis H can be constructed while the following conditions hold:

Necessity:	B	$\not\models$	E^+
Sufficiency:	$B \wedge H$	$\models$	E^+
Weak consistency:	$B \wedge H$	$\not\models$	$\square$
Strong consistency:	$B \wedge H \wedge E^-$	$\not\models$	$\square$

The symbols used above are: $\wedge$ (logical and), $\vee$ (logical or), $\models$ (logically proves), and $\square$ (falsity).

In general, each customer is hosted on a virtual fiber channel fabric for the sake of security (i.e., to achieve isolation) and to provide a redefinable boundary. When a customer encounters problems, they call the support desk and register a problem ticket. Each problem ticket contains the description of the problem, such as accessibility, security issues, performance problems, and so on, along with a suspected set of problematic elements, such as servers, and volumes. Based on the problem tickets, each virtual fabric is marked as a positive or a negative example with respect to time. Problem tickets also help us mark the faulty components at a fine granularity in the fabric based on the customer's report. This step can potentially generate noisy and faulty data. PGML has a module that deals with the noise, but it is currently a work in progress and is beyond the scope of this paper.

PGML performs supervised learning on the attributes defined by the CIM/SMI-S entity and its attributes in order to uncover candidate policies. ILP is well suited for this domain because it has native constructs regarding *entities*, *relationships* between entities, and *logical rules*. This layer defines the generic background and domain knowledge. The CIM/SMI-S model is hierarchical and object oriented. It has constructs such as *CIMClass, CIMInstance, CIMProperty, Aggregation and Association*. *Aggregation* is a particular case of *Association* representing containment. *Association* can be one-to-one, one-to-many, many-to-one, and many-to-many. Background knowledge created by this layer contains information such as how homogeneous or heterogeneous the aggregation set is, the set count, and the cardinality count for the association instances. Each *CIMInstance* has a unique key called *CIMObjectpath* that helps in creating uniqueness check rules. The number of members in a zone (i.e., the zone member count) and whether all servers in a zone have the same or different operating systems is the information that gets retrieved from the association between a fiber channel zone and the fiber channel server ports in it. With this generic, domain-specific background knowledge, the PGML engine internally uses the Progolem ILP tool and orchestrates the flow of data and control across the components described above to generate the hypothesis. These generated hypotheses are the policies or best practices that are then evaluated by the field experts.

V. Evaluation

The goals of our evaluation were to validate the feasibility of a framework that uses ILP-based machine learning over multi-dimensional, relational system management data. In particular we want to: (i) validate PGML generated output with the observations of the field experts, and (ii) evaluate the performance of PGML in terms of sensitivity—to gain insights into the parameters that can affect the efficiency of

our tool, and affect the quality of the best practices that are generated.

Best practice generation: First we provide insights into how PGML generates best practices by transforming the raw data derived from an operational environment into a format interpretable by the machine learning engine. Problem tickets describing the configuration problems that were provided as input to PGML were manually grouped into categories by field experts. Upon generation of best practices, we observed that the generated best practices could be grouped into the same five categories.

We consider an environment E, that consists of elements $e_1, \ldots, e_n$, each of which has attributes $a_1, \ldots, a_t$. Further, we have associations A, with instances $A_1, \ldots, A_k$ where each association A_1 groups a subset of entities in E. Based on the constructs of the basic model shown in Figure 1, we assumed the following concrete model:

```
E =
  {ComputerSystem, StorageVolume, FCPort,
   ProtocolEndpoint, ...}
ComputerSystem =
  {Name, Status, Dedicated, ...}
StorageVolume =
  {DeviceID, BlockSize, Access, ...}
A =
  {SystemDevice, ActiveConnection, ...}
SystemDevice =
  {ComputerSystem, StorageVolume, ...}
ActiveConnection =
  {ProtocolEndpoint, ProtocolEndpoint,
   TrafficType, IsUnidirectional, ...}
```

With the background knowledge, which are the set of possible terms that are provided in the concrete model and could potentially be used as factors of the hypothesis, injected into PGML, the five meta-categories of best practices generated were as follows.

Cartesian: Given a set of values $v_1, \ldots, v_m$ for attributes $a_1, \ldots, a_m$, avoid configurations in which an element e_i belonging to E satisfies

$$\bigwedge_{j=1}^{m} e_i.a_j = v_j.$$

For example, avoid all HBAs of Vendor A type B that do not have firmware versions f1 or f2. A sample of a generated hypothesis under this category was

```
san_configuration(A) :-
  uses_subsystem_model(A, hba01),
  uses_operating_system(A, solaris).
```

Connectivity: Given an association A_i, avoid configurations in which the number of instances of the association A_i between two entities e_a and e_b does not exceed a certain threshold k. For example, avoid all configurations in which a host does not have at least two network paths to a storage subsystem.

Exclusion: Given sets of values $v_{11}, \ldots, v_{1m}$ and $v_{21}, \ldots, v_{2m}$ for attributes $a_1, \ldots, a_m$, avoid configurations of elements e_i and e_j belonging to E that satisfy

$$\Big(\bigwedge_{k=1}^{m} e_i.a_k = v_{1k}\Big) \wedge \Big(\bigwedge_{k=1}^{m} e_j.a_k \neq v_{2k}\Big).$$

For example, tape libraries should not exist in a zone if it contains disk storage controllers.

Many-to-one: Avoid configurations in which the value of some set of attributes $a_1, \ldots, a_m$ is not the same for all entities e_i in an instance of an association A_k. For example, all HBAs associated with a host computer should be from the same vendor with same model and firmware version.

One-to-one: Avoid configurations in which the value of some set of attributes $a_1, \ldots, a_m$ is not different and unique for all entities e_i in an instance of an association A_k. For example, all ports in a storage network fabric must have a unique port world-wide name (WWN).

The generated best practices for each category were then confirmed from hands-on experience with multiple in-house and commercial tools used by administrators today.

Performance evaluation: Next we present performance evaluation results when using PGML. Since the best practice generation is an off-line procedure, it is more important for the system to handle large amounts of learning data, rather than minimizing the response time in terms of best practice generation.

Our experimental setup involves injection of SAN configurations and associated problem tickets into the learning system. Each SAN configuration is considered a data point with two groups of attributes: (i) size (ii) whether it is a positive or negative case. SAN configurations are classified into three broad categories based on their size: (i) small SANs with 25 to 50 connected ports; (ii) medium SANs with 100 to 500 connected ports; and (iii) large SANs with 1000 to 3000 connected ports. Presence or absence of problem tickets associated with a given SAN determine whether it is a positive or a negative data point. Each problem ticket also belongs to one of the five categories that were described earlier. In our overall dataset, 30% of the SANs had problem tickets associated with them, which resulted in a ratio of $\frac{70\%}{30\%}$ of positive to negative SAN data points for PGML.

To measure the sensitivity of PGML to SAN size, we limited our attention to SANs that (possibly) had problems of the Cartesian type. As can be seen in Figure 5, we had a total of 100 small SAN configurations out of which about 30 SANs had problem tickets of the Cartesian type associated with them. Remaining 70 small SANs in the dataset did not have any problem tickets associated with them. Again, as shown in Figure 5, the number of problem reports that were required to generate the best practice dropped from 100 for problem reports from a small SAN size to 20 for those from a large SAN size. The statistical classification results are shown in Table I. This indicates that a large SAN has a greater diversity of information, which leads to improved accuracy in PGML.

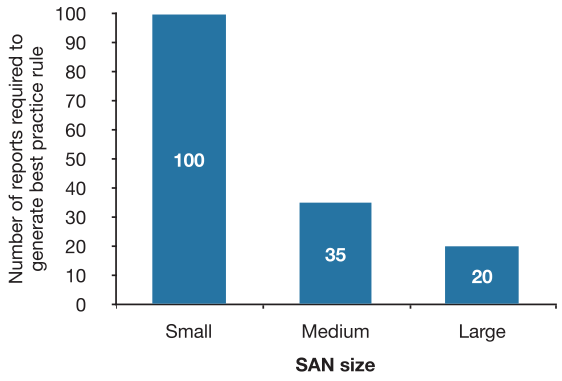

Figure 5. Sensitivity of PGML to size of the SAN in terms of best practice generation

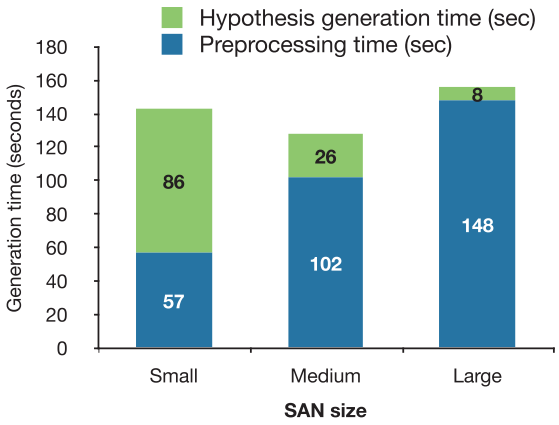

Figure 6. End-to-end run time

Table I
STATISTICAL CLASSIFICATION RESULTS

SAN size	# SANs	Precision	Recall	Accuracy	F-measure
Small	100	0.98	0.97	0.98	0.974
Medium	35	0.92	0.97	0.975	0.944
Large	20	0.88	0.96	0.975	0.918

For the same scenario, we also observed end-to-end run times. Run time was divided into two categories: (i) data preprocessing time, which is the time required to prepare the data so that it can be injected into the ILP engine, and (ii) ILP hypothesis generation time, which is the time taken by the ILP engine to generate hypotheses. Figure 6 shows the time spent in preprocessing of data to generate an input for the ILP engine as compared to the hypothesis generation time. This provides the insight that, given a choice for selection of SAN configurations, we should focus on smaller sets of large SAN configurations: (i) accuracy of analyses over a few large SAN configurations is almost same as over large numbers of small SAN configurations, and (ii) the ILP hypothesis generation engine is more efficient and scalable for fewer large SANs keeping in mind that preprocessing is a prior transformation task that can be parallelized.

VI. Conclusions and Future Work

Best practices are a useful tool in reducing data center management costs while increasing efficiency and ROI (Return On Investment). This is because most configuration problems turn out to be caused by the violation of particular best practices in the storage network domain. The paper presents PGML, a tool that automates the process of generating best practice rules. The tool uses a combination of industry-standard data models, and ILP-based machine learning to statistically infer the best practices for SAN configuration problems.

Future work in this area involves studying applicability to other domains and observing whether the pattern of best practices for storage area networks also is applicable to those domains. Another potential area of work being explored is to integrate information from configuration change logs into our approach and measure the effects on best practice generation. Yet another potential effort would be to measure the impact of domain knowledge on the quality of the best practice rules generated in the domain of storage area networks. One of the areas of research that we are actively pursuing is to measure the sensitivity of PGML to potential misreporting of SAN configuration problems by problem tickets, e.g. real-world problem reports may contain errors due to customers not appreciating the entire problem scope, or due to help-desk imprecision. Our analysis of field data so far shows that this is not an uncommon occurrence, although a precise quantification of this phenomenon is difficult.

This work demonstrates that Machine Learning techniques, carefully applied, can make a useful contribution to the generation of best practice policies within large-scale storage infrastructures.

References

[1] "Amazon Simple Storage Service (S3)," www.amazon.com/s3.

[2] Amazon, "Amazon Elastic Compute Cloud (EC2)," http://www.amazon.com/ec2/.

[3] "iTricity," http://www.itricity.nl/vmwarecloud/.

[4] SNIA, "SNIA Dictionary," http://www.snia.org/education/dictionary/z/.

[5] D. Agrawal, J. Giles, K.-W. Lee *et al.*, "Policy-based validation of SAN configuration," in *POLICY '04: Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2004.

[6] D. Eyers, R. Routray, R. Zhang *et al.*, "Towards a middleware for configuring large-scale storage infrastructures," in *MGC '09: Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science*, 2009.

[7] M. Steinder and A. S. Sethi, "A survey of fault localization techniques in computer networks," *Science of Computer Programming*, vol. 53, 2004.

[8] G. Hamerly and C. Elkan, "Bayesian approaches to failure prediction for disk drives," in *Proceedings of 18th ICML*, Jun. 2001.

[9] E. Kiciman and Y.-M. Wang, "Discovering correctness constraints for self-management of system configuration," in *Proceedings of 1st IEEE ICAC*, May 2004.

[10] I. Steinwart, D. Hush, and C. Scovel, "A classification framework for anomaly detection," *Journal of Machine Learning Research*, vol. 6, Mar. 2005.

[11] P.-N. Tan, S. Michael, and K. Vipin, *Introduction to Data Mining*. Addison Wesley, 2006.

[12] I. Rish, M. Brodie, and N. Odintsova, "Real-time problem determination in distributed system using active probing," in *Proceedings of 9th IEEE/IFIP NOMS*, Apr. 2004.

[13] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox, "Ensembles of models for automated diagnosis of system performance problems," in *Proceedings of DSN*, Jun. 2005.

[14] Y.-M. Wang, C. Verbowski, J. Dunagan, Y. Chen, H. J. Wang, C. Yuan, and Z. Zhang, "Strider: A black-box, statebased approach to change and configuration management and support," in *17th USENIX LISA*, Oct. 2003.

[15] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang, "Automatic misconfiguration troubleshooting with peerpressure," in *Proceedings of 6th USENIX OSDI*, Dec. 2004.

[16] ——, "Why PCs are fragile and what we can do about it: A study of windows registry problems," in *DSN*, Jun. 2004.

[17] K. El-Arini and K. Killourhy, "Bayesian detection of router configuration anomalies," in *Proceedings of ACM SIGCOMM Workshop on Mining Network Data*, Aug. 2005.

[18] K. Nagaraja, F. Oliveria, R. Bianchini, R. P. Martin, and T. D. Nguyen, "Understanding and deailing with operator mistakes in internet services," in *Proceedings of 6th USENIX OSDI*, Dec. 2004.

[19] M. K. Aguilera, P. Reynolds, and A. Muthitacharoen, "Performance debugging for distributed system of black boxes," in *Proceedings of 19th ACM SOSP*, Oct. 2003.

[20] I. Cohen, Z. Steve, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox, "Capturing, indexing, clustering, and retrieving system," in *Proceedings of 20th ACM SOSP*, Oct. 2005.

[21] M. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem determination in large, dynamic internet services," in *Proceedings of DSN*, Jun. 2002.

[22] M. Chen, A. X. Zheng, J. Lloyd, M. Jordan, and E. Brewer, "Failure diagnosis using decision tree," in *Proceedings of 1st IEEE ICAC*, May 2004.

[23] A. Beygelzimer, M. Brodie, S. Ma, and I. Rish, "Test-based diagnosis: Tree and matrix representations," in *Proceedings of 9th IFIP/IEEE IM*, May 2005.

[24] C. J. Chiang, G. Levin, Y. M. Gottlieb *et al.*, "An Automated Policy Generation System for Mobile Ad Hoc Networks," in *POLICY '07: Proceedings of the Eighth IEEE International Workshop on Policies for Distributed Systems and Networks*. Italy: IEEE Computer Society, 2007.

[25] IBM, "IBM Tivoli Storage Productivity Center," http://www-03.ibm.com/systems/storage/software/center/index.html.

[26] EMC, "Ionix ControlCenter," http://www.emc.com/products/storage_management/controlcenter.jsp.

[27] H. Packard, "Hewlett Packard Systems Insight Manager," http://h18002.www1.hp.com/products/servers/management/hpsim/index.html.

[28] S. Gopisetty *et al.*, "Automated planners for storage provisioning and disaster recovery," *IBM Journal Of Research and Development*, vol. Storage Technologies and Systems Vol. 52, no. 4/5, 2008.

[29] ——, "Evolution of storage management: Transforming raw data into information," *IBM Journal Of Research and Development*, vol. Storage Technologies and Systems Volume 52, no. 4/5, 2008.

[30] "Storage Management Initiative Specification (SMI-S)," http://www.snia.org/forums/smi/tech_programs/smis_home/.

[31] DTMF, "Common Information Model (CIM)," http://www.dmtf.org/standards/cim.

[32] ——, "Web Based Enterprise Management (WBEM)," http://www.dmtf.org/standards/wbem.

[33] P. Sarkar, R. Routray, E. Butler *et al.*, "SPIKE: Best Practice Generation for Storage Area Networks," in *SysML '07: Proceedings of the USENIX Second Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, Cambridge, MA, USA, 2007.

[34] A. Srinivasan, "Aleph," http://web.comlab.ox.ac.uk/activities/machinelearning/Aleph/.

[35] S. Colton, "The HR Program for Theorem Generation," in *Proceedings of CADE'02*, Copenhagen, Denmark, 2002.

[36] S. H. Muggleton, "Progol," http://www.doc.ic.ac.uk/~shm/progol.html.

[37] ——, "Inductive logic programming," *New Generation Computing*, vol. 8, no. 4, pp. 295–318, 1991.

[38] S. H. Muggleton and L. de Raedt, "Inductive logic programming: Theory and methods," in *Proceedings of Journal of Logic Programming*, 1994, pp. 629–679.

[39] S. H. Muggleton, "Inductive Logic Programming: derivations, successes and shortcomings," *SIGART Bulletin*, vol. 5, no. 1, pp. 5–11, 1994.