Automated storage network capacity management utilizing simulation and optimization

Michael O'Sullivan and Cameron Walker

{*michael.osullivan, cameron.walker@auckland.ac.nz*}
Department of Engineering Science
University of Auckland
Auckland 1142, New Zealand

Abstract

There is a wealth of research on storage systems, focusing on the storage system architecture, intelligent use of storage resources, and mass storage systems. However, very little research looks at a key component of centralized storage systems, the physical fabric of the network that holds the storage system together. "Best practice" frameworks for the management of storage systems exist, but the design of the network fabric within storage systems has relied heavily on the knowledge and expertise of storage systems architects, administrators and managers.

In this paper we present the Network Capacity Management Cycle (NCMC), a new framework for network capacity management. This framework is unique because it was developed around well-established Operations Research methods, namely simulation and optimization. Applying these methods to the network fabric of storage systems allows a majority of the framework to become automated, significantly reducing the workload for storage systems architects, administrators and managers.

The NCMC utilizes cutting-edge network discovery and monitoring tools, leading network simulation software and new methods for network capacity design. We describe each of the steps of the NCMC in detail and discuss how to automate almost all of these steps. A case study is also presented that demonstrates one iteration of the NCMC applied to an existing storage system in the Department of Engineering Science at the University of Auckland. In addition to illustrating the steps of the process, this case study also outlines the numerous complications we have encountered in our initial use of the NCMC.

1 Introduction

The exponential expansion in the quantity of information requiring storage has made the design and management of storage systems and their associated storage networks an increasingly important component of the digital infrastructure that organizations rely on to function effectively. As the scale of the storage systems increase in response to growing data needs, the design of the storage network fabric needed to support these systems is becoming too complicated to deal with manually. In Alvarez et al.'s [AKM+00] comprehensive tutorial on storage systems management they recognize both the importance and difficulty of storage network fabric design – which they refer to as storage area network (SAN) fabric design – and described the state of the art at the time as "designs done by hand, using a few simple topologies".

There is a wealth of innovative research on storage systems, particularly into the design of the storage architecture [GNA+98, GFS03, KW03], the allocation of storage resources [ABG+01, ASS+05], the simulation of SANs [MSSD00, WZZ+03, CFDL08] and the design of mass storage systems [GSK03, HM04]. However, there is a dearth of research into the design of the actual network, also known as the interconnect fabric, used with storage systems. In many cases [GFS03, KW03, WZZ+03] the interconnect fabric is represented as a "cloud" or high-speed network without any further discussion. In others, the fabric is not considered at all [ABG+01, ASS+05], the fabric design is given a priori [GNA+98, MSSD00, GSK03, CFDL08] or a number of set fabric designs are considered [HM04].

Due to the computational complexity of the problem, storage network fabric design tools that are currently available run coarse heuristic algorithms

on rough estimates of data flow [WOSW02, Str02]. Even if these tools could include accurate profiles of storage network traffic, this information is seldom known by storage network administrators, and difficult to come by.

Over the last five years the authors' primary area of research has been the automatic design and reprovisioning of storage networks using integer programming techniques [OW05, WOT07, WO08]. However, when trying to implement these techniques on existing storage systems, the absence of a rigorous strategic framework for managing the storage network fabric within storage systems has been apparent. The lack of fundamental mathematical modeling procedures, incorporating accurate network traffic profiles, to understand and predict the performance of this essential fabric has lead the authors to conceive the Network Capacity Management Cycle (NCMC), a framework for utilising Operations Research methods to automate the management of the storage network fabric. This framework could be used within management frameworks for the entire storage system (see, for example, [AKM+00, Fuj, Inf]) or the NCMC could also be employed as a management framework for entire storage systems by incorporating system parameters and decisions into the appropriate steps of the NCMC. In this paper we describe the phases of the NCMC and present a case study of its application.

The rest of this paper is structured as follows. In section 2 we present the NCMC and outline the detail in each of its steps. In sections 3 to 7 we discuss our experience applying each of the steps of the NCMC to a storage system in the Department of Engineering Science at the University of Auckland, New Zealand. In section 8 we conclude our study and discuss future directions.

2 The Network Capacity Management Cycle

The Network Capacity Management Cycle (NCMC) is a framework to ensure that the design, management, and ongoing reprovisioning of a network infrastructure follows sound practice and is based on quantifiable objectives. It is based strongly on modeling practices in Engineering Design, whereby accurate monitoring of a system provides data that is used in an iterative cycle. When applied to storage network fabric management, the uniqueness of the NCMC derives from the embedded Operations Research methods. The NCMC framework includes the development, calibration and validation of multiple simulation models and the formulation and solution of an optimization model.

The steps of the NCMC are shown below in figure 1. The detail in these steps is explained in the rest of this section.

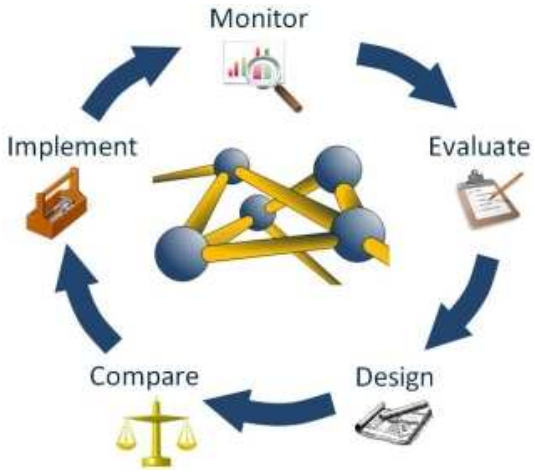

Figure 1: The Network Capacity Management Cycle

2.1 Monitor the Network

This "step" of the NCMC is really a continuous process involving the monitoring of the storage network infrastructure and traffic transmitting across the network. Accurate profiles of network traffic and component failure are maintained for use in the subsequent steps of the NCMC. When starting the NCMC, the monitoring step may also require network discovery, whereby the physical layout of the network components is determined. Thenceforth the layout is documented in a network diagram which can be updated as changes are made.

2.2 Evaluate the Network

In this step storage network administrators evaluate the current network fabric (find bottlenecks, test failures, etc). A diagram of the network layout together with current traffic profiles should be used to build a simulation model of the network. Scenarios of possible future usage patterns and network failures should then be tested on this model, to determine how the network will perform under possible traffic loads. Another innovative approach that may be employed in this step is the use of virtual machines and emulated networks [CFDL08] to model the current system.

2.3 Design the Network

Here, storage network administrators use the information gained from step 2.1 (*Monitor the Network*) and step 2.2 (*Evaluate the Network*) to design an optimal network. Optimal network design is an NP-hard optimization problem, and has been the main focus of our research to date. Currently we use integer programming to perform this step [WOT07, OW05], although some heuristic algorithms do exist [WOSW02, Str02].

2.4 Compare the Networks

In this step storage network administrators compare two network layouts in terms of performance, reliability, etc. This can be achieved by using the same methods in step 2.2, that is, simulation or emulation, to model the new network design (from step 2.3) and running the current traffic profiles and the future usage scenarios (also from step 2.2) across this model. If the comparison is not favorable the modeling in the design step should be investigated and an improved design produced.

2.5 Implement the Network

If a decision is made to replace the current network fabric with the new design, the new network fabric is implemented, tested and deployed in place of the existing network. Our expectation is that the NCMC will produce a new fabric design that can replace the original network fabric without adversely affecting the existing storage system architecture. However, it may be that either a new storage architecture has been suggested as one of the future usage scenarios or the new fabric design requires a new storage architecture to function. In both of these cases the new architecture will have been incorporated into the comparisons from step 2.4 and the storage network administrators will have approved the architectural change.

The NCMC is intended as a framework for the ongoing management of a storage network fabric. If the network under consideration is monitored regularly iterations of the cycle can be triggered by significant changes, either in existing traffic profiles, or in expected future usage patterns, or in the architecture of the enveloping storage system.

The NCMC can be embedded as a subprocedure within existing storage systems management framework [AKM+00, Fuj, Inf]. For example, the first step in Fujitsu's Network Life Cycle Services is "Plan" and within that step is the "Network design" task that involves "Design of network architecture and topologies, traffic, capabilities and performance". When performing this task storage system architects could use the NCMC, but instead of implementing the final network design they could pass this design into the next task of the "Design" step, "Interoperability testing".

Storage systems architect could also use the NCMC as their management framework, but incorporate extra system parameters into steps 2.1, 2.2, 2.4 and extra design decisions into step 2.3. For example, in step 2.3 (*Design the Network*), the acquisition and allocation of storage resources (using a tool such as MINERVA [ABG+01]) could be the first task followed by the design of a storage network fabric (using the methods discussed in section 5) to support these storage resources. As another example, simulation products such as OPNET Modeler[1] can explicitly model the protocols and applications within a storage system, thus allowing for different storage architectures to be deployed across the same storage network fabric.

For the remainder of this paper we assume the NCMC is only being used to for the management of the storage network fabric and the storage architecture, application servers and clients, storage devices, etc have all been designed and deployed. However, during our discussion of the implementation of a storage network fabric (see section 7) we do refer to our experience deploying a storage architecture.

In order to understand the practical complexities inherent in the steps of the NCMC framework we undertook to perform one iteration of our framework to redesign an existing network. As relative novices in the fields of network engineering and storage systems, a hands-on implementation of our management framework was essential to appreciate the viability of our framework as standard best-practice for storage network fabric management. It is our intention to automate the steps of the NCMC as much as possible, an objective that has become a major focus of our future research in the short term.

3 Monitoring a Network

The Department of Engineering Science is one of five departments within the School of Engineering at the University of Auckland, New Zealand. Engineering Science has approximately 30 permanent staff and a student population of just under 200 stu-

[1] OPNET Modeler is a product of OPNET Technologies, Inc., see http://www.opnet.com/solutions/network_rd/modeler.html for details.

dents. At the beginning of our prototype study we knew that both staff and students had data storage available as "floating" drives via the Windows XP operating system on the departmental network. We were also aware that some staff and students also used data storage on the Linux operating system, but we were unsure how this data was accessed.

The monitoring step of the Network Capacity Management Cycle involves measuring how the storage network is performing. In later iterations of the cycle this should involve measuring data traffic in the storage network. However, in the initial iteration of the NCMC, the storage network itself must be discovered. During our prototyping within Engineering Science, we discovered a "disconnection" between the people who use the network (the IT staff within Engineering Science) and the people who maintain the network, in this case Information Technology Services (ITS), the central IT administrators at the University of Auckland. The Engineering Science IT staff knew which machines were connected to the network, but only had a vague idea what the internal network topology was. ITS had a good idea of the internal network topology (though no network diagram), but had only a vague idea about the machines connected to the network and the applications and protocols running across the network. This disconnection between the network administrators and network users is not uncommon. For this reasons we chose to utilize network discovery tools to determine the overall topology of network.

The use of network discovery tools for the monitoring step of the NCMC predominantly automates this step. In our prototyping we had to request permission for SNMP access to the switches on the network but nothing more. We experimented with two discovery tools: LANsurveyor[2] and LanTopolog[3]. LANsurveyor uses multiple protocols to discover the network topology (for example ICMP Ping, NetBIOS, SIP, etc) whereas LanTopolog automates the discovery of the physical network using SNMP. During our prototyping, LANsurveyor discovered the internal structure of the network, but, on many switches, could not identify the machines connected to the switch ports. LanTopolog successfully discovered the entire network, but it's port labeling was rudimentary and it only supplied MAC addresses for the machines connected to the network. While neither of these tools was completely satisfactory, since LanTopolog discovered the entire network we could combine its layout map with a mapping of MAC addresses to IP addresses[4] to complete our picture of the network topology.

The topology is shown in figure 2. The data storage is supplied by 3 servers, all shown on the right:

1. Andromeda – the Windows server that stores the Engineering Science staff data;

2. Leopard – the Windows server that stores the Engineering Science student data;

3. Engsci-linux – the Linux server that stores both staff and student data.

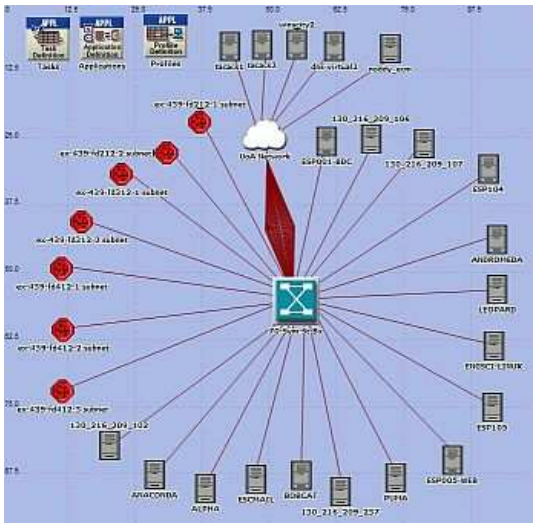

Figure 2: Engineering Science Network Topology

These servers have two "hot swappable" disks each in a mirrored configuration to avoid data unavailability in case of disk failure. However, if any of these servers fails then all data of that server is unavailable until it is repaired or replaced.

Once the network topology has been successfully determined during the monitoring step in the initial iteration of the NCMC a network diagram can be created. Any further changes to the network should be added to the network diagram or else the network topology will need to be rediscovered.

Given that the network topology is known, the main function of the monitoring step is to capture the usage of the network. There are many applications that capture traffic on a network including tcp-

[2]LANsurveyor is a product of SolarWinds, see http://www.solarwinds.com/products/lansurveyor/ for details.

[3]LanTopolog is freeware from Yuriy Volokitin, see http://lantopolog.googlepages.com/ for details.

[4]This mapping was provided by GFI LANguard Network Security Scanner 8.0, a product of GFI Software, see http://www.gfi.com/lannetscan/ for details.

dump[5], Wireshark[6] and ACE[7]. Using one of these applications we can capture the way applications use the storage network. In our prototyping research we selected tcpdump after their were some network congestion issues with the ACE capture agents.

All three applications use WinPcap/LibPcap and their capture files can be easily visualized with ACE. Figure 3 shows a *Tier Pair Circle* for a network capture run on Engsci-linux on the Engineering Science network. The Tier Pair Circle shows different conversations observed by Engsci-linux. Also, once inside ACE the capture data can be deployed within a network simulation (see section 4 for details).

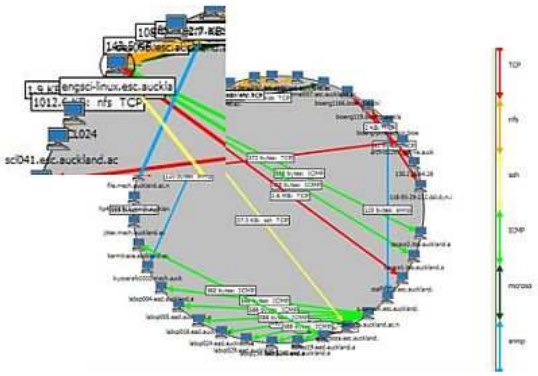

Figure 3: Tier Pair screenshot from ACE (the location of Engsci-linux is enlarged)

4 Evaluating a Network

There are 3 ways to evaluate a network: build the network and test it; simulate the network; and emulate the network. In the Network Capacity Management Cycle the network already exists, but is in use, so testing is not desirable. Both simulation and emulation can be time consuming, but with the network topology and traffic already known from the monitoring step (see section 3), it should be relatively straightforward to build and validate a (simulation or emulation) model of the existing network. In our prototyping research we initially experimented with generic simulation packages [WOE05, Son07] before turning to specialist products such as ns-3[8] and OPNET Modeler. We selected OPNET Modeler because of its ease of use and the existence of the eXpress Data Import (XDI) module that allowed switches in the network to be automatically loaded into a simulation model. New research into the emulation of TCP networks [CFDL08] also shows promise as a largely automated evaluation tool, but we have not explored this avenue at this time.

Using OPNET Modeler we created a simulation model of the Engineering Science network, a screen shot of the model is shown in figure 2. We also deployed the network captures (from tcpdump) as traffic in the simulation model using ACE combined with Modeler. Early results are promising, although a full validation of the network simulation still remains to be done.

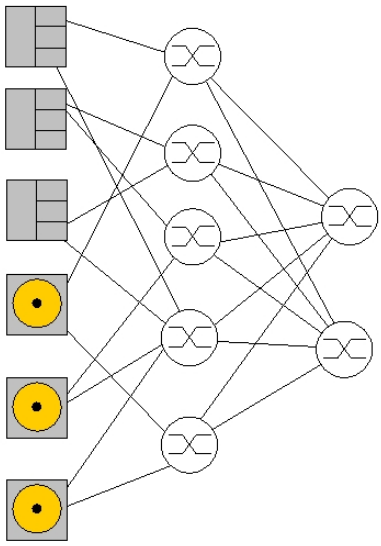

Figure 4: A Single-edge Core-Edge Design

5 Designing a Network

The design step of the Network Capacity Management Cycle has been the major focus of our research to date. Our initial design have used a Core-edge topology because of its inherent reliability and scalability. The Core-edge topology consists of two disjoint cores of high-bandwidth switches connected to one or two layers of lower-bandwidth edge switches.

The edge switches connect to the hosts and devices in a Single-edge (see figure 4) or Double-edge design[9] Under the Core-edge topology, no host-device

[5]tcpdump, and its Windows equivalent Windump, are freeware often distributed with operating systems, see http://www.tcpdump.org/ for details.

[6]Wireshark is freeware sponsored by CACE Technologies, see http://www.wireshark.org/ for details.

[7]ACE Modeler is a product of OPNET Technologies, Inc., see http://www.opnet.com/solutions/ application_performance/acelive.html for details.

[8]ns-3 is freeware designed for internet systems, see http://www.nsnam.org/index.html for details.

[9]In this paper we restrict our designs to the Single-edge topology, but our design methods are easily extended to produce

pair is separated by more than three intermediate connections (hops). Determining a design with two disjoint paths between every host-device pair is simplified by the disjoint nature of the cores. Furthermore, with the addition of switches to the edges or the core, an existing design can be scaled to accommodate additional hosts or devices.

Previously we have modeled the Core-edge SAN Design Problem (CESANDP) as a mixed-integer program (MIP) [WOT07], including explicit connection of servers and storage devices (unlike existing heuristic solutions for Core-edge design) and an over-subscription ratio. Figure 5 depict the decisions in our mixed-integer programming formulation:

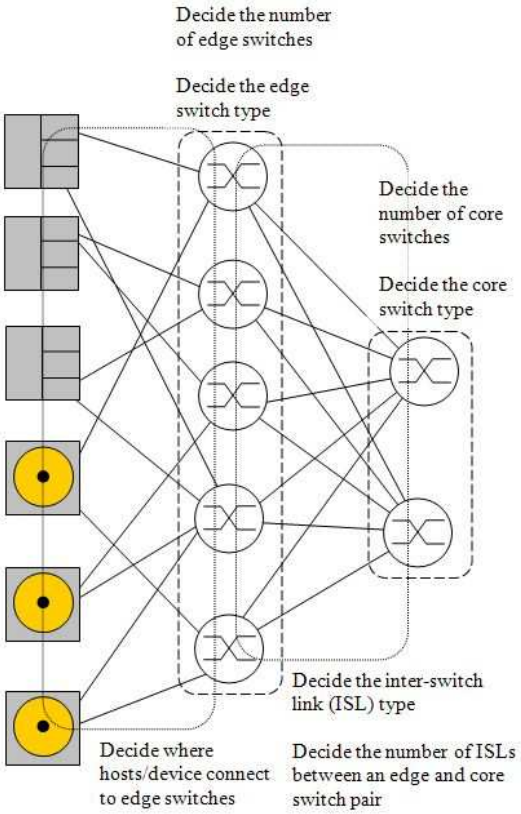

Figure 5: Decisions for a Single-edge Core-Edge Design.

- The number of core switches;
- The type of switch used for *all* core switches, i.e., every core switch will be of the same switch type;
- The number of edge switches;
- The type of switch used for *all* edge switches;
- The number of inter-switch links (ISLs) between each edge and core switch, i.e., there are the same number of ISLs between every core-edge pair;
- The type of link used for *all* ISLs;
- To which edge switch(es) each host/device connects (usually, ≥ 2 for reliability).

Our recent research has yielded improvements to our MIP formulation [WO08]. We present here a summary of that formulation.

5.1 Problem Description

First, we introduce our notation and present the problem description for a Single-edge CESANDP. In this problem, we need to design a network with a single layer of switches at the edge and a layer of switches as the core. The network provides full pairwise connectivity between the application servers and/or client machines, denoted as a set of hosts H, and the data storage, denoted as a set of devices D.

Each host $h \in H$ (device $d \in D$) has a fixed number of ports, $\eta(h)$ (respectively $\eta(d)$), a port bandwidth capacity, $\beta(h)$ (respectively $\beta(d)$), and port cost $\pi(h)$ (respectively $\pi(d)$). The objective of the CESANDP problem is to determine the design of least cost network fabric that fully connects the hosts and the devices.

The Core-edge topology requires a number of design restrictions:

1. There is a constant number of links, and a single link type, between each edge switch and each core switch;
2. All switches in the edge must be of a single switch type and all core switches must be of a single type;
3. To ensure performance, each edge switch port has a greater bandwidth capacity than its incident link to a core link;
4. The bandwidth capacity of any edge link incident to a host h (device d) must be greater than $\beta(h)$ ($\beta(d)$, respectively). The same is true for the port bandwidth of the edge switch port that link connects to;
5. The network is built to support a minimum number, ζ, of disjoint paths between every host-device pair.

We include an over-subscription ratio at the edge switches in our design, as is standard in industry.

Double-edge networks.

The over-subscription ratio κ is the ratio of the instantaneous bandwidth capacity for data flowing into the edge switch (from the hosts or devices) to the instantaneous bandwidth capacity for data flowing out of the edge switch (toward the core).

To build the network fabric we use sets of different switch types $\mathcal{S}$ and link types $\mathcal{L}$. Each switch type $t \in \mathcal{S}$ has an associated cost $\gamma(t)$ and port cost $\pi(t)$. Similarly each link type $u \in \mathcal{L}$ has cost $\gamma(u)$. The number of ports on a switch of type t can be decided by the user, up to an upper bound $\overline{\eta}(t)$, and each switch has a port bandwidth upper bound of $\overline{\beta}(t)$. Each link type u has a bandwidth upper bound $\overline{\beta}(u)$.

5.2 The CESANDP mixed-integer programming formulation

In this section we summarize the mixed-integer programming (MIP) formulation for solving the (Single-edge) CESANDP.

5.2.1 Switch and Link Sets

We define three sets of switches:

- *edge* switches S^E;
- *core* switches S^C;
- the set of all switches $S = S^E \cup S^C$.

We also define the set of hosts, devices and switches as the network *nodes*, $N = S \cup H \cup D$.

We define two sets of links:

- edge links, L^E, from the hosts and devices to the edge switches;
- core links, $L^C = \cup_{e \in S^E, c \in S^C} L^{(e,c)}$, where $L^{(e,c)}$ is the set of links between edge switch $e \in S^E$ and core switch $c \in S^C$.

Each link $l \in L$ has a source $\sigma(l) \in N$ and a destination $\tau(l) \in N$. The source and destination nodes for each set are restricted: $\sigma(l) \in H \cup D, \tau(l) \in S^E$ for $l \in L^E$; $\sigma(l) = e, \tau(l) = c$ for $l \in L^{(e,c)}$.

5.2.2 Decision Variables

The decision variables determine which links and switches to include in the network fabric. The set of switches S is *generic*, so we must decide the type of each switch:

$$x_{s,t} = \begin{cases} 1 & \text{if switch } s \text{ is type } t; \\ 0 & \text{otherwise} \end{cases}, s \in S, t \in \mathcal{S}$$

Note, not every switch must be assigned a type, some switches may not be included in the fabric. Also, the number of ports of switch type $t \in \mathcal{S}$ on switch $s \in S$ is given by $n_{s,t}$.

All edge switches that are used must be the same type, likewise for core switches:

$$e_t = 1 \text{ if switch type } t \text{ used for edges}, 0 \text{ otherwise}$$
$$c_t = 1 \text{ if switch type } t \text{ used for cores}, 0 \text{ otherwise}$$

Like the switches, the links are generic, but the cardinality of the links is also unknown. We define two sets of variables for the links between the hosts/devices and edge switches:

$$y_l = \begin{cases} 1 & \text{if } l \text{ is used in the fabric} \\ 0 & \text{otherwise} \end{cases}, l \in L^E$$
$$k_{l,u} = \text{the number of links of type } u \text{ along } l, \quad l \in L, u \in \mathcal{L}$$

To determine the links between the edge switches and core switches we define:

$$m = \text{the number of links between each edge switch and core switch}$$
$$z_u = \begin{cases} 1 & \text{if we use link type } u \text{ between} \\ & \text{the edge and core switches} \\ 0 & \text{otherwise} \end{cases}, u \in \mathcal{L}$$

5.2.3 Objective Function

Our aim of the CESANDP is to minimize the cost of the storage network fabric, that is the total cost of switches, switch ports and links used in the design fabric.

$$Z = \sum_{s \in S, t \in \mathcal{S}} \underbrace{(\gamma(t) x_{s,t} + \pi(t) n_{s,t})}_{\text{switch and port cost}} + \sum_{l \in L, u \in \mathcal{L}} \underbrace{\gamma(u) k_{l,u}}_{\text{link cost}}$$

Note that using a host (device) port incurs a cost $\pi(h)$ ($\pi(d)$), but, as the design fabrics are fully connected, these costs are omitted from the objective function.

5.2.4 Ordering and Antisymmetry Constraints

One common problem for network design formulations is symmetry amongst solutions. To remove symmetry from the solution space we order both the edges switches and the core switches, and impose a hierarchy of utilization:

$$x_{s+1,t} \leq x_{s,t}, s \in S^E, t \in \mathcal{S}, \quad (1)$$
except the last edge switch

$$x_{s+1,t} \leq x_{s,t}, s \in S^C, t \in \mathcal{S}, \quad (2)$$
except the last edge switch

Note that $s+1$ denotes the switch after s in the same set.

5.2.5 Link Constraints

Using the existence variables for the links $y_l, l \in L^E$, we ensure the bandwidth of links between hosts (devices) and edge switches is larger than the port bandwidth of hosts (devices):

$$\sum_{u \in \mathcal{L}} \beta(u) y_l \geq \beta(\sigma(l)), l \in L^E$$

We also make sure that the existence variables "control" to the link cardinality variables appropriately, and vice versa:

$$\sum_{u \in \mathcal{L}} k_{l,u} \leq M y_l, l \in L^E \quad (3)$$

$$y_l \leq \sum_{u \in \mathcal{L}} k_{l,u}, l \in L^E \quad (4)$$

$$\sum_{l \in L^E, \tau(l)=s} y_l \leq M \sum_{t \in \mathcal{S}} x_{s,t}, s \in S^E \quad (5)$$

$$\sum_{t \in \mathcal{S}} x_{s,t} \leq \sum_{l \in L^E, \tau(l)=s} y_l, s \in S^E \quad (6)$$

The big-*M* quantity in (3) is the minimum of the maximum number of ports on a switch, and the maximum number of ports on a host or device. The big-*M* quantity in (5) is the minimum of the maximum number of ports on a switch, and the total number of hosts and devices.

5.2.6 Switch Constraints

Similarly to (3)-(4), the switch existence variables control the number of ports variables, and vice versa:

$$n_{s,t} \leq \overline{\eta}(t) x_{s,t}, s \in S, t \in \mathcal{S} \quad (7)$$

$$n_{s,t} \geq x_{s,t}, s \in S, t \in \mathcal{S} \quad (8)$$

The number of ports used on each an edge switch is also bound by the edge switch type, similarly for the core switches.

$$n_{s,t} \leq \overline{\eta}(t) e_t, s \in S^E, t \in \mathcal{S} \quad (9)$$

$$n_{s,t} \leq \overline{\eta}(t) c_t, s \in S^C, t \in \mathcal{S} \quad (10)$$

Lastly, the number of ports on a switch is equal to the number of links connected to it.

$$\sum_{\substack{u \in \mathcal{L} \\ l \in L^E, \tau(l)=s}} k_{l,u} + \sum_{\substack{v \in \mathcal{L} \\ j \in L^C, \sigma(l)=s}} k_{j,v} = \sum_{t \in \mathcal{S}} n_{s,t}, s \in S^E \quad (11)$$

$$\sum_{\substack{u \in \mathcal{L} \\ l \in L^C, \tau(l)=s}} k_{l,u} = \sum_{t \in \mathcal{S}} n_{s,t}, s \in S^C \quad (12)$$

Similar to (5.2.5), we ensure the ports of each edge switch provide greater bandwidth than the port bandwidth of any connected hosts or device:

$$\beta(\sigma(l)) y_l \leq \sum_{t \in \mathcal{S}} \overline{\beta}(t) e_t, l \in L^E \quad (13)$$

Finally, we ensure that the hosts and devices are fully connected:

$$\sum_{\substack{u \in \mathcal{L} \\ l \in L^E, \sigma(l)=h}} k_{l,u} = \eta(h), h \in H \quad (14)$$

$$\sum_{\substack{u \in \mathcal{L} \\ l \in L^E, \sigma(l)=d}} k_{l,u} = \eta(d), d \in D \quad (15)$$

5.2.7 Oversubscription Constraints

To accurately model the over-subscription we define a new set of variables for the edge switches:

$f_s =$ the total possible data flow out of $s, s \in S^E$

and use this to restrict the bandwidth of links to the edge switch:

$$\sum_{\substack{u \in \mathcal{L} \\ l \in L^C, \tau(l)=s}} \overline{\beta}(u) k_{l,u} \leq \kappa f_s, s \in S^E \quad (16)$$

The data flow out of an edge switch is restricted by the port bandwidth of the edge switch, the bandwidth of the core-edge link and the port bandwidth

of the core switch:

$$f_s \leq \overline{\beta}(t) \sum_{\substack{u \in \mathcal{L} \\ l \in L^{(e,c)}}} k_{l,u} + M(1 - e_t), \quad e \in S^E, t \in \mathcal{S} \tag{17}$$

$$f_s \leq \sum_{\substack{u \in \mathcal{L} \\ l \in L^{(e,c)}}} \overline{\beta}(u) k_{l,u}, e \in S^E \tag{18}$$

$$f_s \leq \overline{\beta}(t) \sum_{\substack{u \in \mathcal{L} \\ l \in L^{(e,c)}}} k_{l,u} + M(1 - c_t), \quad e \in S^E, t \in \mathcal{S} \tag{19}$$

The big-*M* quantity we use in (17) and (19) is the sum of the port bandwidths across all hosts and devices.

5.2.8 Architecture Constraints

The architecture constraints must enforce a single edge switch type:

$$\sum_{s \in S^E} x_{s,t} \leq M e_t, t \in \mathcal{S} \tag{20}$$

$$x_{s,t} \geq e_t - (1 - \textstyle\sum_{u \in \mathcal{S}} x_{s,u}), \quad t \in \mathcal{S}, s \in S^E \tag{21}$$

$$\sum_{t \in \mathcal{S}} e_t = 1 \tag{22}$$

a single core switch type:

$$\sum_{s \in S^C} x_{s,t} \leq M c_t, t \in \mathcal{S} \tag{23}$$

$$x_{s,t} \geq c_t - (1 - \textstyle\sum_{u \in \mathcal{S}} x_{s,u}), \quad t \in \mathcal{S}, s \in S^C \tag{24}$$

$$\sum_{t \in \mathcal{S}} c_t = 1 \tag{25}$$

and a single link type between each edge and core switch:

$$\sum_{l \in L^C} k_{l,u} \leq M z_u, u \in \mathcal{L} \tag{26}$$

$$k_{l,u} \geq z_u - (2 - \textstyle\sum_{t \in \mathcal{S}} (x_{e,t} + x_{c,t})), \quad u \in \mathcal{L}, l \in L^{(e,c)} \tag{27}$$

$$\sum_{u \in \mathcal{L}} z_u = 1 \tag{28}$$

and the same number of links between each edge and core switch:

$$\sum_{u \in \mathcal{L}} k_{l,u} \leq m, l \in L^{(e,c)} \tag{29}$$

$$\sum_{u \in \mathcal{L}} k_{l,u} \geq m - M(2 - \textstyle\sum_{t \in \mathcal{S}} (x_{e,t} + x_{c,t})), \quad l \in L^{(e,c)} \tag{30}$$

The big-*M* quantity in (20) is the cardinality of S^E and in (23) it is the cardinality of S^C. Note, in (21) and (24), if switch s is not used $(1 - \sum_{u \in \mathcal{S}} x_{s,u})$ is equal to 1 and these constraints are void.

The big-M quantity in (26) is the maximum number of ports on any switch multiplied by either the maximum number of edge switches or core switches allowed – whichever quantity is the smaller of the two. The big-M quantity in (30) is simply the maximum number of ports on any switch. Note, in constraints 27 and 30, if either switch e or c is not used $(2 - \sum_{t \in \mathcal{S}} (x_{e,t} + x_{c,t}))$ is greater than 0 and these constraints are void.

5.2.9 Minimum Path Constraints

When building the Core-edge fabric we must ensure we provide (at least) the right number of paths ζ through the network:

$$\sum_{\substack{s \in S^E \\ t \in \mathcal{S}}} x_{s,t} \geq \zeta \tag{31}$$

$$\sum_{\substack{s \in S^C \\ t \in \mathcal{S}}} x_{s,t} \geq \zeta \tag{32}$$

$$\sum_{l \in L^E, \sigma(l) = h} y_l \geq \zeta, h \in H \tag{33}$$

$$\sum_{l \in L^E, \sigma(l) = d} y_l \geq \zeta, d \in D \tag{34}$$

Here, we have also included some basic cuts to speed up the solution process – the number of connections from one host or device to any single edge can be at most the number of ports on the host or device, minus $\zeta - 1$.

$$\sum_{\substack{l \in L^E, \sigma(l) = h, \tau(l) = s \\ u \in \mathcal{L}}} k_{l,u} \leq \eta(h) - (\zeta - 1), \quad h \in H, s \in S^E \tag{35}$$

$$\sum_{\substack{l \in L^E, \sigma(l) = d, \tau(l) = s \\ u \in \mathcal{L}}} k_{l,u} \leq \eta(d) - (\zeta - 1), \quad d \in D, s \in S^E \tag{36}$$

5.2.10 Preprocessing and Cut Generation

In [WO08] we also experimented with an optimization-based preprocessing step for selecting the best switch or link for a given set of properties, for example, the maximum number of ports and port bandwidth for switches. We also created a number of problem specific cuts that speed up the solution time of the MIP using properties of the

CESANDP, for example, the total number of host and device ports. For more detail please refer to [WO08].

5.3 Prototyping the Design step

During our prototyping research we designed a Core-edge storage network as an alternative to the direct attached storage currently being used (see section 3). First, we added a backup server that was in the process of being added to the existing network. Next, we added two servers to act as clustered storage controllers. We removed the disks from the original storage servers (Andromeda, Leopard and Engsci-linux respectively) and combined their storage into 2 external storage devices to allow mirroring.

We assumed there would be at most 10 edge switches and 5 core switches (this would provide a very overprovisioned network fabric). We wanted to support the full bandwidth of the hosts and devices so we used an over-subscription of 1 (that is, no over-subscription) and we requested at least 2 paths between any host and any device. The full model has 318 variables and 941 constraints. The variables consist of 212 binary variables and 106 integer variables. We use the commercial MIP solver software AMPL/CPLEX[10] to solve our design problem.

We solved our model on a Dell Precision M4300 laptop with an Intel Core 2 Duo CPU T9500 with two 2.6 GHz processors and 3.5 GB of RAM. The LP relaxation was solved by CPLEX 9.1.0 in 0.03 seconds, with the full MIP formulation solving in 0.312 seconds. Our preprocessing and innovative cuts combined with the cuts generated by CPLEX to solve the MIP from the root node with branching.

The simulation model for the solution to the CESANDP formulation is shown in figure 6.

This solution provides at least two disjoint paths between every host-device pair, but the formulation does not model traffic profiles across the network explicitly. Instead, the over-subscription rate is included to ensure no component is too heavily loaded. We do have MIP formulations that explicitly include traffic profiles in the design process, and are currently developing specialized procedures (for example, column generation) procedures to give solutions in a reasonable time-frame.

On last consideration, the cost of ILOG's commercial licenses for AMPL/CPLEX may price them beyond the reach of storage network administrators, so we are currently developing automatic design capabilities in COIN's open source Branch Cut and Price (BCP) package[11]. Furthermore, BCP will allow us more control in directing the solution process.

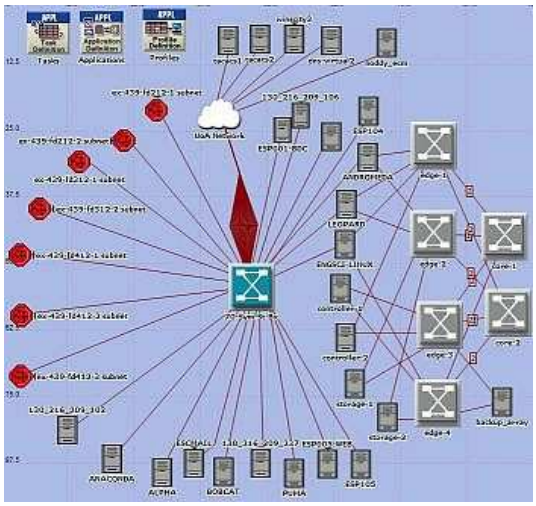

Figure 6: Core-Edge Network Topology

6 Comparing Networks

Once a new storage network fabric design has been produced (in the design step, see section 5) the Network Capacity Management Cycle compares the two designs to ensure the newly designed fabric performs better than the existing storage network fabric. This performance comparison may include scenarios such as the current network traffic, future usage predictions, component failures within the fabric, etc. The current network has already been either simulated or emulated in the evaluate step (see section 4), so the network topology produced by the design methods (see section 5) must also be either simulated or emulated. It is not compulsory to use the same approach used in section 4, but it simplifies comparisons.

In our prototyping research we built a simulation of the new Core-edge storage network fabric in OPNET Modeler. Figure 6 shows a screenshot of this model. With the two simulation models (of the existing network – figure 2 – and the new network – figure 6) constructed, we initially deployed estimated network usage on both networks. The results showed that the new fabric design shared the load amongst the servers and also load-balanced across

[10] AMPL/CPLEX is a product of ILOG, see http://www.ilog.com/products/ for details.

[11] BCP is part of the COIN-OR project, see http://www.coin-or.org/projects/Bcp.xml for details.

the fabric. Moreover, the new design showed comparable performance to the existing network when one of the components of the storage system had failed.

At the time of writing we are continuing to monitor the existing network and will use the actual traffic scenarios to test both networks with real-world usage. Analysis of these traffic captures may also provide insight into "typical" usage and allow us to forecast traffic patterns for the future. Using these forecasts we can also compare how the two networks will perform in years to come.

7 Implementing Networks

Given the promising initial results from the comparison step (see section 6), and our intention to complete one iteration of the NCMC, we have been developing the storage network designed by our optimization (see section 5). Before we have done significant testing, we will deploy this network into the Engineering Science IT systems. To perform this testing we purchased the two servers for the storage controller cluster and built two servers to act as the external disks with 2TB of disk space each. We have also purchased 4 ethernet switches (two to act as edge switches and 2 to act as core switches). The total cost of the network was less than US$7000. In comparison the cost to build a similar system using a Dell PowerVault MD3000 (with 2TB of disk space) is over US$12000. Of course, there are extra costs involved with getting a "home-brew" system into production, but the initial savings are very promising.

To further prove our fabric design, we used open source software to implement the storage architecture. We built the storage controller cluster using the Red Hat Cluster Suite and presented the disks to the cluster over the network using Open-iSCSI and iSCSI Enterprise Target and software RAID. We have built a Red Hat Global File System system on storage for use by servers connecting to the storage controller cluster. Our next step is to test the performance of this system using Iometer and monitor the behavior of the cluster, GFS, iSCSI and software RAID for incorporation into our network simulation. We are very close to demonstrating that a storage network created by the NCMC can be successfully implemented.

8 Conclusions and Extensions

The NCMC is unique, not as a framework from storage systems management, but because of its focus on the storage networking fabric and its embedded Operations Research methods. In this paper we have described the steps of the NCMC in detail. Our research has shown us that the NCMC is a promising framework for storage networking fabric management. It produces low cost fabric designs that are able to support a novel storage architecture. Adding a little "glue" between the steps of the NCMC will allow the entire framework to become a largely automated process that will save storage architects, administrators and managers a lot of time and effort.

Using freely available network discovery and monitoring tools, the monitoring step of the NCMC can provide a detailed picture of the storage network being managed, and its performance, with very little user effort. Then, in the evaluation a virtual model of the existing network is quickly built in a state-of-the-art simulation package and future use scenarios or rigorous testing may be done without affecting the actual network. Next, the design step automatically produces a new, optimized storage network fabric. The comparison step also quickly produces a virtual model of the new design and the usage scenarios from the evaluation step can be swiftly migrated into this new simulation model. The implementation step of the NCMC requires the most work from the storage architects, administrators and managers. However, this step is probably the easiest for experienced storage systems practitioners and they will be confident in their new system given the underlying Operations Research tools utilized for strategic decisions. Further development of effective storage architectures that can be deployed over the designs produced by the NCMC will speed up the implementation step. While users of the NCMC will need to build their knowledge of the tools being employed, once they gain experience the NCMC will be relatively straightforward and largely automated. The Network Capacity Management Cycle represents an exciting new evolution in storage networking capacity management.

References

[ABG+01] Guillermo A. Alvarez, Elizabeth Borowsky, Susie Go, Theodore H. Romer, Ralph Becker-szendy, Richard Golding, Arif Merchant, Mirjana Spasojevic, Alistair Veitch, and John Wilkes. Minerva: an automated resource provisioning tool for large-scale storage systems. *ACM Transactions on Computer Systems*, 19, 2001.

[AKM+00] Guillermo Alvarez, Kim Keeton, Arif Merchant, Erik Riedel, and John Wilkes. Storage systems management. SIGMETRICS '00 Tutorial, June 2000.

[ASS+05] Eric Anderson, Susan Spence, Ram Swaminathan, Mahesh Kallahalla, and Qian Wang. Quickly finding near-optimal storage designs. *ACM Trans. Comput. Syst.*, 23(4):337–374, 2005.

[CFDL08] Carlo Caini, Rosario Firrincieli, Renzo Davoli, and Daniele Lacamera. Virtual integrated tcp testbed (vitt). In *TridentCom '08: Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*, pages 1–6, ICST, Brussels, Belgium, 2008.

[Fuj] Fujitsu Computer Systems. Fujitsu Network Life Cycle Services. http://www.fujitsu.com/us/services/telecom/services/.

[GFS03] J. S. Glider, C. F. Fuente, and W. J. Scales. The software architecture of a san storage control system. *IBM Systems Journal*, 42(2):232–249, 2003.

[GNA+98] Garth A. Gibson, David F. Nagle, Khalil Amiri, Jeff Butler, Fay W. Chang, Howard Gobioff, Charles Hardin, Erik Riedel, David Rochberg, and Jim Zelenka. A cost-effective, high-bandwidth storage architecture. *SIGOPS Oper. Syst. Rev.*, 32(5):92–103, 1998.

[GSK03] Gregory R. Ganger, John D. Strunk, and Andrew J. Klosterman. Self-*:storage : Brick-based storage with automated administration. Technical Report CMU–CS–03–178, Computer Science Department, School of Computer Science, Carnegie Mellon University, August 2003.

[HM04] Andy Hospodor and Ethan L. Miller. Interconnection architectures for high-performance file systems. In *Proceedings of the 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2004)*, pages 273–281, College Park, MD, April 2004.

[Inf] Information Technology Infrastructure Library. The ITIL Core Framework. http://www.itil.org/en/.

[KW03] Kimberly Keeton and John Wilkes. Automatic design of dependable data storage systems. In *Proceedings of Workshop on Algorithms and Architectures for Self-managing Systems*, pages 7–12, San Diego, CA, June 2003.

[MSSD00] Xavier Molero, Federico Silla, Vicente Santonja, and José Duato. Modeling and simulation of storage area networks. *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2000. Proceedings. 8th International Symposium on*, pages 307–314, 2000.

[OW05] Michael O'Sullivan and Cameron Walker. A mixed-integer approach to storage area network design using generic network components. Technical report, Department of Engineering Science, University of Auckland, New Zealand, 2005.

[Son07] D. Soni. Simplifying the simulation of computer switches using regression. Master's thesis, University of Auckland, Auckland, NZ, 2007.

[Str02] S. Strand. Storage area networks and santk. Master's thesis, University of Minnesota, Minnesota, USA, 2002.

[WO08] Cameron Walker and Michael O'Sullivan. Core-edge design of storage area networks – a single-edge formulation with problem-specific cuts. Submitted to Operations Research, 2008.

[WOE05] Cameron Walker, Michael O'Sullivan, and Mihiriyani Elangasinghe. Evaluation of core-edge storage area network designs using simulation. Technical report, Department of Engineering Science, University of Auckland, New Zealand, 2005.

[WOSW02] Julie Ward, Michael O'Sullivan, Troy Shahoumian, and John Wilkes. Appia: automatic storage area network design. In *Conference on File and*

Storage Technology (FAST'02), pages 203–217, January 2002.

[WOT07] Cameron Walker, Michael O'Sullivan, and Timothy Thompson. A mixed-integer approach to core-edge design of storage area networks. *Computers and Operations Research*, 34(10):2976–3000, 2007.

[WZZ+03] Chao-Yang Wang, Feng Zhou, Yao-Long Zhu, Chong Tow Chong, Bo Hou, and Wei-Ya Xi. Simulation of Fibre Channel Storage Area Network Using SANSim. In *The 11th IEEE International Conference on Networks (ICON 2003)*, September 2003.