Appia and the HP SAN Designer: automatic storage area network fabric design

Julie Ward, Michael O'Sullivan[1], Troy Shahoumian, John Wilkes, Ren Wu, Dirk Beyer

jward@hp.com, michael.osullivan@auckland.ac.nz, troy_shahoumian@hp.com, wilkes@hpl.hp.com, ren_wu@hp.com, dirk_beyer@hp.com

HP Laboratories
[1] University of Auckland, Auckland, New Zealand

Abstract

Designing a SAN fabric involves finding a network of hubs, switches and links to connect hosts to their storage devices. The network must be capable of simultaneously meeting the data flow requirements between multiple host-device pairs, and it must do so cost-effectively, since large-scale SAN fabrics can cost millions of dollars. Moreover, the network must be robust to failures of fabric elements. Given that the data flows can number in the hundreds, simple over-provisioned manual designs are often not attractive; they cost significantly more than they need to, may not meet the performance needs, may expend valuable resources in the wrong places, and are subject to human error.

Producing SAN fabric designs automatically can address these difficulties, but it is a non-trivial problem; it extends the NP-hard minimum-cost fixed-charge multicommodity network flow problem to include degree constraints, node capacities, node costs, unsplittable flows, and other requirements. Nonetheless, we have developed two efficient algorithms for automated SAN fabric design. We give an overview of these algorithms and compare their performance over a range of design problems. We also discuss how they have been extended to reprovision existing SAN fabrics. Finally, we describe how, through the creation of the HP SAN Designer tool, these algorithms have been transferred to the HP SAN design community with the expert help of the Network and Storage Support Services Organization.

1 Introduction

A SAN (storage area network) consists of a group of servers (or hosts) connected to shared storage devices (such as disks, disk arrays and tape drives) through an interconnection fabric consisting of switches, hubs, and links. An example of a SAN is shown in Figure 1, with hosts represented in the top row of components, devices in the bottom row, and switches and hubs in between. SANs offer many advantages over direct-connected local storage, including superior connectivity of servers to storage devices, better utilization of storage resources, centralized administration and management, increased scalability, and improved performance.

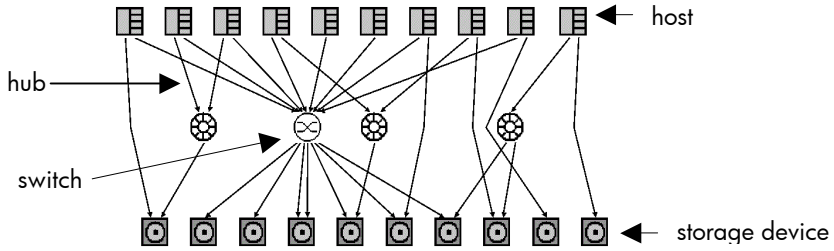

Figure 1. A simple, single-layer SAN fabric. Hosts appear in the top row, devices in the bottom row, and switches and hubs in between.

In spite of SANs' many advantages, their adoption is hampered by the complexity of designing them. Designing even a small SAN fabric requires considerable time and effort from IT experts, whose manual methods often result in expensive, over-provisioned designs. This problem intensifies as the designs get larger and more complex. The SAN fabric can comprise 10-20% of the total storage system cost, and SAN designs that require

millions of dollars of fabric alone are becoming more common. We have witnessed a factor of three difference in the cost of a SAN between a manual design ($4m) that took several days and an automatically-generated one ($1.4m) that took a few minutes.

Design mistakes can be subtle and therefore easy to overlook, yet potentially very costly; poor performance is commonplace, and downtime in failure situations can result if the failure-tolerance aspects are mis-designed. As SANs grow to include hundreds or even thousands of storage devices, it becomes increasingly difficult, even for SAN experts, to manually design cost-effective and reliable SANs.

We believe the solution to these problems is to automate SAN design. An automated method must account for the performance requirements (to avoid queuing or packet loss), and should try to minimize system cost, because SAN components are typically quite expensive. Automated design would enable wider deployment of SANs, and increase the likelihood that the systems deployed would meet real needs.

We have developed just such a solution: a tool to design SANs automatically. We call it Appia, after the Appian Way, part of the network of roads leading to ancient Rome. Appia includes two algorithms developed at HP Labs for cost-effective and reliable SAN fabric design. These two approaches, which we call *Flowmerge* and *Quickbuilder*, demonstrate complementary strengths. *Flowmerge*, which is more computationally intensive, tends to find lower-cost designs for SANs with sparse connectivity requirements, whereas *Quickbuilder* excels when connectivity requirements are dense. We found that the better of the two designs is, on average, within 33% of the optimal design cost for empirical test problems that are small enough to solve optimally. Moreover, these designs are found in a few minutes or less for SANs with 50 hosts and 100 devices, a size typical of the largest current installations. Appia's algorithms are also capable of reprovisioning existing SAN fabrics to avoid costly, error-prone disruptions to the existing network infrastructure. Appia has been transferred to storage field the through the creation of the HP SAN Designer, a joint effort between HP Labs and the Storage Support Services Organization (N3SO).

2 The SAN fabric design problem

The SAN fabric design problem can be stated quite simply: we are given a set of hosts, a set of storage devices, and a set of requirements in the form of data flows between host-device pairs. Each flow has a desired bandwidth. The goal is to build a minimum-cost SAN fabric to support all of these requirements simultaneously. To do so, one must select a set of fabric nodes (switches and hubs) from available types, a set of links connecting pairs of nodes (hosts, devices and fabric nodes), a topology with which to join these together, and a single path through the network for each flow. (The single-path restriction arises from SCSI request-ordering constraints.)

The resulting fabric design must be *feasible* - that is, it must satisfy constraints that ensure it is buildable, and it must support the connection and performance requirements. These constraints are: (1) the number of links connected to a host, device or fabric node must not exceed the number of ports available there (these restrictions are called *degree constraints*) and (2) the flow routing must honor the bandwidth limitations of links and fabric nodes. Because packets travel differently through hubs and switches, their bandwidth constraints differ. Packets routed into a switch are forwarded directly to the next destination in their path. In contrast, packets routed into a hub are transmitted through all connected hubs and all links attached to these hubs; they are seized by their next destination. Thus, the total flow into an interconnected set of hubs is limited by the minimum of the bandwidth of each individual hub, the bandwidth of each connected link, and the bandwidth of each port used by these links. The bandwidth of switches is therefore more efficiently utilized than hub bandwidth.

In addition to the constraints described above, it is also important that a SAN fabric provide redundancy to avoid access interruptions when fabric elements fail. The Flowmerge and Quickbuilder algorithms, as they are described in sections 3 and 4, build non-redundant fabrics. However, section 6 describes how these algorithms have been extended to provide two different types of reliability.

2.1 Related work

Currently, SAN design is done manually by IT experts. For small designs, they typically use drag-and-drop interfaces, an error-prone approach that makes it hard to include performance requirements. For larger SANs, the alternative is using canned topologies that often result in grossly over-provisioned designs. While over-provisioning can be advantageous, it is important that it is done strategically to provide high performance, scalability, reliability, and robustness to changes in requirements. One canned design used commonly today is the Brocade Core-Edge architecture [4]. Such standardized topologies are used when the SAN designers have no systematic way to predict the connectivity and data flow requirements in their SANs, and so opt for full connectivity between hosts and devices. But this flexibility comes at a very high price. In general, when any information is available about SAN requirements, far more cost-effective designs can be found.

As part of our search for algorithms to apply to this problem, we turned to the literature in network design. Unfortunately, most traditional network design approaches minimize only link costs, because switches are cheaper than trenching in wide area telephone networks, which are the target of most of this work. In the SAN case, the opposite is usually the case: a fully loaded 64-port FibreChannel "storage director" switch costs close to $200,000, while individual fibre links cost a few hundred dollars. As a result, the existing research in network design proved less applicable than we had hoped.

The SAN fabric design problem generalizes and extends several NP-hard problems in the network design literature, including the multicommodity network design problem [8,3,2,7,9,5,1], which involves choosing a minimum-cost set of capacitated, fixed-cost links to connect a known set of nodes in order to satisfy multicommodity flow requirements. This problem is known to be NP-hard even in the single commodity case, and is notoriously difficult to solve in practice. Unlike in multicommodity network design, the nodes of a SAN fabric are not known *a priori*; instead, they must be selected from a wide variety of types of hubs and switches, differing in attributes such as cost, bandwidth, and number of available ports. This difference makes the search space of the SAN fabric design problem considerably larger than conventional network design problems. Moreover, SAN fabric design further generalizes the multicommodity network design problem to include degree constraints, node capacities, node costs, and unsplittable flows.

The many features of the SAN design problem have been addressed individually or in small subsets in previous work. (See [11] for a more detailed literature review.) The first to address all of its features in a common framework was an algorithm called *Merge*, also developed at HP Labs [10]. *Merge* found cost-effective designs for small problems but failed to find feasible designs for larger problems. The algorithms presented here are proven to find feasible designs under a reasonable set of conditions, and their designs are generally more cost-effective than those found by *Merge*.

3 The *Flowmerge* algorithm

This section provides an overview of the *Flowmerge* algorithm. *Flowmerge* is named for the way it "merges" individual flows into sets of flows that share a hub, switch or link. It was motivated by the observation that when two flows are routed together along a link from host or device they have in common, a port on that host or device is conserved. *Flowmerge* attempts to alleviate the shortage of host and device ports by selecting subsets of flows with common hosts or devices, and routing them together through shared links.

Flowmerge is a recursive algorithm that creates a SAN design by introducing, at each recursive application, a set of fabric nodes and links, with no links between fabric nodes in the set. When the algorithm terminates, the fabric design consists of one or more "layers" of nodes, where there are links between but not within layers. An example of a layered fabric produced by *Flowmerge* is shown in Figure 2. The top and bottom rows of components contain hosts and devices, respectively, and the remaining components are fabric nodes.

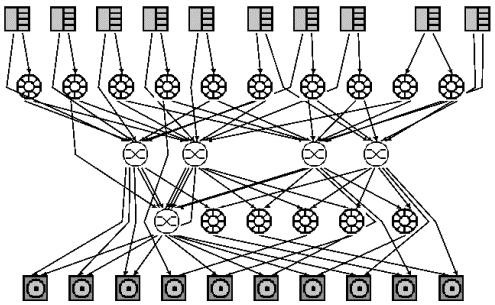

Figure 2. A sample SAN fabric produced by *Flowmerge*

The process through which one layer of fabric is introduced is called *Single-Layer Flowmerge*. The input to *Single-Layer Flowmerge* is a set *H* of hosts, a set *D* of devices, and flow requirements *F* between them. *Single-Layer Flowmerge* produces a series of single-layer fabric designs to support the flow requirements. Each design in the series is feasible with respect to all constraints except, possibly, the degree constraints on hosts and devices. The initial design consists of a direct host-device link for each flow. This design is typically infeasible because one or more hosts or devices has fewer ports than incident links. The difference between the numbers of incident links and available ports on a given host or device is called its *port violation*. Each subsequent design in the series has a smaller total port violation than the previous design, or a lower cost than the previous design if both designs are feasible.

To see how this series of designs is obtained, consider an arbitrary single-layer fabric. Associated with each fabric node in the design is a subset of flow requirements routed via that node. Similarly, associated with each direct host-device link in the fabric is a subset of flows routed along that link. In general, the flow requirements are partitioned into disjoint subsets, such that each flow requirement is in exactly one subset. Each subset in the partition has an associated fabric node or direct host-device link through which all flows in the subset are routed. We call these subsets *flowsets*.

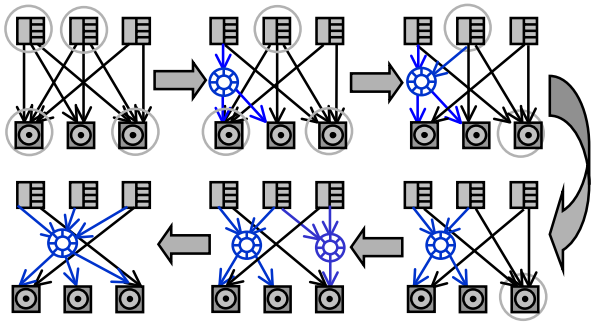

Figure 3. Example application of *Single-Layer Flowmerge*. The problem has 3 hosts and 3 devices, each with 2 ports, and a single type of switch available with 8 ports. The eight flows in the problem each have bandwidth 33 MB/s. Links and ports have bandwidth 100 MB/s. Six successive designs are shown, beginning with the one that assigns each flow to its own link. In each design, hosts and devices with the highest port violation are circled. For example, in the first design, the highest port violation is one: there are two hosts and two devices each with three incident links and only two ports. Each design in the series reduces the port violation on one host or device from the previous design by merging two flowsets together. After four mergers, all port violations are eliminated. The last merge eliminates one fabric node and thereby reduces the cost of the fabric.

Single-Layer Flowmerge begins with the finest partition of the flow requirements: each flow is in its own flowset. At each iteration, a new, coarser, partition is obtained by merging two flowsets together. When merging two flowsets, we must select a fabric node type among available types with which to route the flows in the merged

flowset, and the links connecting hosts and devices to the node along which we route the flows. The node type is selected based on the number of ports available on the node and the cost of using the node (including the cost of required ports and links). We select the flowsets to merge to alleviate port violations, favoring reductions on the hosts and devices with the most severe violations. Cost is a tie-breaker criterion. Once two flowsets are merged, they are never split. *Single-Layer Flowmerge* continues merging flowsets until either no two flowsets can be merged, or all port violations have been eliminated and no merger produces a cost savings. *Single-Layer Flowmerge* terminates, because after a finite number of mergers (one less than the number of flows, at most) only a single flowset remains, so no further mergers are possible. Figure 3 demonstrates how *Single-Layer Flowmerge* works on a small example.

A single layer of fabric nodes is often not enough to alleviate all port violations on hosts and devices. When this is true, *Single-Layer Flowmerge* is reapplied recursively to generate cascading layers of fabric nodes. With the introduction of each layer, host and device port violations are reduced. For a more detailed description of how *Flowmerge* creates multiple layers of fabric, please see [11].

4 The *Quickbuilder* algorithm

Our second algorithm for SAN fabric design is called *Quickbuilder*. It was motivated by the following observation: since flows cannot be split across multiple paths in the SAN fabric, each flow must be assigned to a single port on its host and device. The assignment of flows to ports significantly impacts the remainder of the SAN design. Specifically, a given port assignment creates a partition of the host and device ports into disjoint subsets of ports called *port groups*. The port group of port p is a set of ports that includes p; if q is a port in the port group and a flow assigned to q is also assigned to port r, then r is in the port group. In short, the port group of port p includes p, all ports p must communicate with, all ports they communicate with, etc. The critical insight is that each port group can be treated as an independent, smaller design problem. We refer to the fabric required to support a single port group as a *module.* In general, the fewer ports in a port group, the smaller the module is required to support its flows. Thus, a finer decomposition results in a less costly fabric. *Quickbuilder* seeks an assignment that results in a fine decomposition.

Quickbuilder is a two-phased algorithm. In the first phase of the algorithm, the *port assignment* phase, *Quickbuilder* assigns each flow, one at a time, to a single port on its host and a single port on its device. *Quickbuilder* selects the assignment for the flow that results in the lowest estimated cost impact on the fabric. When estimating the cost of a flow's assignment, *Quickbuilder* determines the port groups that would be created by adding the assignment to previous flows' assignments. It then estimates the cost to support each of these port groups with a module. *Quickbuilder* chooses the assignment whose incremental estimated total cost is lowest. It continues making the lowest-cost assignment for each flow until all flows have been assigned to ports.

The assignment obtained in the first phase implies a partition into port groups. The second, *module-building* phase of the algorithm considers each port group created in the port assignment phase separately, and builds a module to support the flows assigned to its ports. *Quickbuilder* seeks the lowest-cost module for each port group. It chooses from one of three different types of modules, depending on the number of ports in the port group and the total bandwidth of flow assigned to it. A direct-link module consists of a single host-device link; it is used only for port groups containing two ports. A hub module is composed of one or more hubs in series, each connected to the next by a single link; hub modules are used whenever a direct-link module is insufficient, but the bandwidth of flows in the port group does not exceed the bandwidth of a hub. The third type of module, a switch-module, is one that contains at least one switch. It is only used when the other types of modules cannot be. This selection criterion is based on the realistic assumption that the cost of a link is less than that of a hub, and both are dramatically less than the cost of a switch.

Two examples of *Quickbuilder* designs are shown in Figure 4 and Figure 5. The fabric in Figure 5 was developed by *Quickbuilder* with the same inputs that *Flowmerge* used to find the fabric in Figure 2. For this problem, *Quickbuilder*'s assignment of flows to ports led to two port groups, one of which is very large, containing all but two ports. The fabric contains one direct host-device link, and one very large module with three interconnected switches. Figure 4 is a solution to the SAN design problem for which *Flowmerge* designed the fabric in Figure 1.

In this fabric, *Quickbuilder*'s port assignment created five port groups. Two port groups are supported by direct links, two larger port groups are supported by hubs, and the largest is supported by two switches connected to each other. In the next section we compare the *Quickbuilder* and *Flowmerge* solutions in more detail.

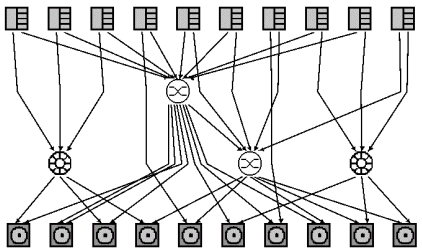

Figure 4. A sample SAN fabric produced by *Quickbuilder* (cf. Figure 1)

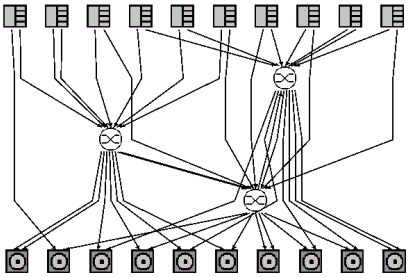

Figure 5: A second *Quickbuilder* SAN fabric (cf. Figure 2)

5 Evaluation of the algorithms

We have no analytical bounds to describe how close the *Flowmerge* and *Quickbuilder* designs are to optimal. However, we have done extensive empirical testing to compare their designs to each other and to optimal designs produced by solving an integer program. This comparison is described in detail in [11]; we summarize it here.

Naturally, the best evaluation of Appia comes from applying its algorithms in a real SAN environment and comparing their designs to alternative designs produced by SAN design experts. This comparison should include several metrics, including fabric cost, performance, availability, scalability and even aesthetics. We have had few opportunities to evaluate Appia algorithms in this context so far, but in those cases, Appia found much cheaper designs than those produced manually. In one such case, a consultant worked for several days to produce a $4 million design on a problem that Appia solved for $1.4 million in a few minutes. The consultant's design used several expensive, 64-port switches, and a completely symmetrical solution; *Flowmerge* found ways to achieve the same goals using much cheaper 16-port switches. Nonetheless, we will refrain from making strong claims about the benefits of our approach until we have had more opportunities to evaluate it on a wider range of real-world problems. The recent release of the HP SAN Designer should provide such opportunities in abundance.

In the meantime, we have created a large set of realistic test problems based on our observations about real-world problems. The problems vary in size (numbers of hosts and devices) and in properties of the flow requirements.

In comparing the two algorithms' effectiveness over these test problems, we found that *Flowmerge* produces lower cost designs than *Quickbuilder* for smaller problems, whereas for large problems, *Quickbuilder* produces dramatically cheaper designs. The relative advantages of *Quickbuilder* can be seen by a direct comparison of the fabrics in Figure 2 and Figure 5, found by *Flowmerge* and *Quickbuilder*, respectively, for the same problem. The *Flowmerge* fabric uses five switches and fifteen hubs, and costs $265,080. *Quickbuilder* produced a $133,440 fabric using only three switches. This problem has a relatively large number of flow requirements per host and device. In such problems, often every possible assignment of flows to ports results in one port group containing all or most of the host and device ports. Thus, problems of this type require a large fabric through which almost all host and device ports are interconnected. *Quickbuilder*'s switch-module building method makes very cost-effective use of switches in this setting.

Flowmerge usually needs multiple fabric layers to connect all ports in a large port group. Its myopia in building independent layers and in performing only pairwise mergers rather than multi-flowset mergers, both without backtracking, impairs its effectiveness in such problems. For example, in Figure 2, the middle layer of fabric was introduced first without regard for how it would affect future layers, and subsequent layers were built independently of the others. Moreover, it overlooked cost-saving multi-flowset mergers in the outermost layers.

Contrastingly, *Flowmerge*'s relative strengths for problems with fewer flows per host and device are apparent

when comparing the fabrics in Figure 1 and Figure 4 for the same problem. *Flowmerge*'s $63,720 fabric uses only one switch and three hubs, whereas *Quickbuilder* produced a more expensive $97,120 fabric. This might be explained by *Quickbuilder*'s quite myopic port assignment method, which ignores the flows that have yet to be assigned while making its current assignment. The port assignment determines the decomposition of ports into port groups, and thereby a decomposition of flows into disjoint subsets that can be routed through independent fabric elements. In this particular example, *Flowmerge*'s fabric has four distinct port groups, the largest containing sixteen ports. *Quickbuilder* created five port groups with twenty-four ports in the largest group. Large port groups typically lead to more fabric elements. *Flowmerge* excels at finding finer decompositions in less dense problems. Thus, *Flowmerge*'s strength is assigning flows to ports in such a way to yield smaller port groups, whereas *Quickbuilder* is better at building modules for large port groups when they are unavoidable. This supports an obvious strategy: run both algorithms, and pick the better solution. When the HP SAN Designer invokes the Appia algorithms for a SAN design, it does exactly that.

In comparing Appia designs to optimal ones produced by an integer program (IP), we could only compare to the smallest problems in our test suite. A problem with 10 hosts and 10 devices has over forty thousand binary variables and seventy five thousand constraints, a size far beyond the capabilities of today's commercial IP solvers. Figure 6 focuses solely on the smallest problems, with five hosts and five devices, because these problems can be solved optimally by the IP. It shows the relationship between the optimal design cost and the cost of the designs produced by the two heuristics. It indicates that for small problems, *Flowmerge* and *Quickbuilder* find solutions that are, on average 38% and 55% over the optimal fabric cost, respectively. Though it is not shown in this graph, a closer inspection of the individual tests reveals that in all of these small tests except for a few in which each host and device has a large number of low bandwidth flows relative to its total port bandwidth, *Flowmerge* and *Quickbuilder* find designs that average within 13% and 25% of the optimal design cost, respectively. The fourth bar contains the cost produced by the linear programming (LP) relaxation of the integer program, created by relaxing integrality constraints on the IP variables. The LP does not produce usable designs, but it does provide a lower bound on the optimal cost. In these small problems, the lower cost bound is, on average, half of the optimal cost.

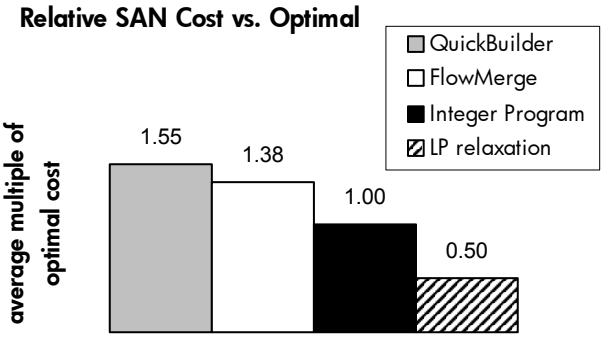

Figure 6: Cost comparisons of the resulting SAN designs for four different design algorithms, averaged across all 5 host, 5 device tests. “Integer program” (IP) produces optimal solutions, *Flowmerge* and *Quickbuilder* are heuristics that produce feasible solutions; LP relaxation produces a guaranteed lower bound, but (in general) infeasible solutions.

In addition to comparing the cost-effectiveness of the designs produced by *Flowmerge* and *Quickbuilder*, it is interesting to compare their relative efficiency. *Quickbuilder* is dramatically faster than *Flowmerge*; for our largest test problems, with 50 hosts, 100 devices and 600 flows, *Flowmerge* finds a design in less than 10 minutes whereas *Quickbuilder* finds one in less than 40 seconds. Both algorithms’ running times are well within the range of practicality for implementation within an interactive tool such as HP SAN Designer.

6 Reliability

It is critical in most business environments that a SAN continues to provide access to data even in the event of failures of fabric elements. The Flowmerge and Quickbuilder algorithms, as they have been described, do not address reliability. However, a simple method for reliable SAN design is to include two copies of a fabric that supports all of the flow requirements. Thus, any algorithm to design a SAN fabric without reliability can be generalized to create such a doubled network, provided a sufficient number of ports are available on hosts and devices. One simply reserves half of the ports available on each host and device, finds a non-reliable design using the remaining ports, duplicates this fabric and attaches the second copy to the reserved host and device ports. The combined fabric provides for each flow two simultaneously available paths that are *non-intersecting*, i.e., they have no common fabric element. It can therefore support all flow requirements even in the event of simultaneous failure of multiple elements in one copy of the fabric.

Because it is rare that more than one network element is non-operational at any given time, it is generally considered sufficient to design SAN fabrics with no single point of failure. A generalization of *Flowmerge* can be used to that end. A rough outline of the method is as follows. The first step is to reserve a single port on each host and device, and build a *primary* fabric with the remaining ports by applying a minor variation of the *Flowmerge* algorithm. This variation ensures that for each host and device, a fabric failure in the primary fabric affects flows from at most one port on that host or device. The second step is to apply a different variation of *Flowmerge* to build a *secondary* fabric with the reserved ports and a duplicate set of flows. This secondary fabric must support the flows impacted by any single fabric failure in the primary network. In this *Flowmerge* variation, flowsets still correspond to flows routed together through a common fabric node, but these flows need not be routed simultaneously. Instead, only the flows affected by the same fabric failure in the primary network need to be supported at once. The design of the secondary fabric takes into account the specific vulnerabilities of the primary fabric. This approach to fault-tolerant SAN design creates two non-intersecting paths for each flow (one in each of the primary and secondary fabrics) without requiring that these paths be simultaneously available.

7 Reprovisioning SAN fabrics

Storage needs inevitably evolve over time. Many SAN designs today are done in an environment in which a SAN already exists and needs to be modified to adapt to changed requirements. Thus, perhaps more important than the problem of designing a SAN fabric from scratch is that of reprovisioning an existing SAN to accommodate changes and additions to hosts, devices and flow requirements. In reprovisioning, one would like to make use of existing fabric elements, and must also avoid unnecessary disruptions to existing connections. Because the wiring of SAN fabrics is typically very complex, recabling requires expensive expert labor and is quite error-prone.

We have created a three-phased method for reprovisioning that leverages both the *Flowmerge* and *Quickbuilder* algorithms. In the first phase of this method, we attempt to route flows in the existing network. If there are flows that cannot be routed, then in the second phase we selectively add links to the network in order to provide paths for unrouted flows. If adding links is insufficient to provide routes for all flows, we apply a more drastic approach in the third phase: we treat the existing hardware as spare parts whose costs are zero, and we solve a new SAN fabric design problem using either of the existing algorithms. Because the algorithms regard the spare parts as free, they favor the reuse of the existing hardware wherever it is practical.

In situations where requirements change dramatically and unpredictably, this approach will result in disruptions to existing wiring patterns. Because recabling is often impractical, we are exploring less disruptive methods for reprovisioning. One promising approach employs integer programming to determine which flows are routed in the existing SAN, and which portions of the existing SAN to preserve. These decisions are made with respect to the combined objectives of maximizing the count or volume of flow routed in the existing SAN, and minimizing the cost of SAN disruptions. For each flow that is not routed through the existing SAN, the integer program also chooses the ports through which that flow exits and reenters the existing SAN. *Flowmerge* or *Quickbuilder* can then be applied to build a new SAN, using each flow's exit and reentry port as its hosts and devices in the new problem. The final step is to combine the new SAN with the preserved portions of the existing SAN. While

integer programming was not a scalable approach for the SAN design problem, the integer program used in this approach has considerably fewer decision variables and thus a much smaller search space. Preliminary experiments suggest that this approach can produce less disruptive re-designs in quite reasonable running times.

8 The HP SAN Designer

Thanks to the leadership and efforts of N3SO, Appia is now the core design engine for HP SAN Designer, a software tool for the HP consulting, support and sales force in storage. Version 1.0 of the HP SAN Designer was released in January 2003.

HP SAN Designer provides HP storage professionals with an assisted way to create, check, display, and report the design of storage area networks.

It provides a graphical user interface (implemented using third party software), through which users can invoke the Appia algorithms for designing and reprovisioning SAN fabrics. In doing so, they can specify their desired level of reliability, and can select the types fabric elements from which the algorithms can choose. The interface also allows users to run a validation algorithm, also developed at HP Labs, to determine whether a SAN fabric meets the performance requirements and is physically buildable. Moreover, it allows them to manually design and modify SAN fabrics. This interactive environment allows the user to combine the results of the algorithms with their own design preferences, and quickly explore the implications of many different input scenarios. Other features of HP SAN Designer include detailed reporting of a SAN's bill of materials, and automated checking against user-specified conformance rules, such as restrictions on numbers of hops or inter-switch links.

In its second release, planned for August of 2003, HP SAN Designer will be integrated into N3SO's SANExpert architecture. SANExpert is a common framework for a suite of tools for analysis, diagnostics, and design of SAN environments. The integration of HP SAN Designer into SANExpert will achieve several important goals. First, because SANExpert includes a Visio-based editor to allow visual display and manual editing of SANs, HP SAN Designer's current dependency on third party software will be eliminated in the second release, thus reducing licensing costs and enabling broader distribution within HP. Second, by providing a centralized architecture through which all of N3SO's SAN tools can communicate, SANExpert will significantly simplify the lives of the storage field. Many other valuable tools will be included in SANExpert, such as: SANDiff, a troubleshooting tool that compares snapshots of a SAN configuration at two different points in time; SANValidate, an engine to check a design against a database of interoperability rules; and SANEditor, the Visio-based graphical interface.

Though it is still too soon to assess the true value of the HP SAN Designer, early reports make us optimistic that it will be a significant asset to the storage field. Beta-testers felt that the tool would save them valuable time in the SAN design process, improve the quality of their designs, and improve the customer experience overall. As just one example of what the tool can do, a senior HP SAN architect reported it "invaluable in ... helping to design new SANs [and] also to double-check existing designs." One test case with 600 nodes had taken a week to generate by hand; the tool was able to produce a comparable solution in less than a day - even during the beta test. For the designer, the big benefits he emphasized were (1) being able to check a design for correctness, and (2) to explore multiple "what if" scenarios – "For example, I can now try out the differences between using director class switches versus all small switches, to see which would make the best configuration for the customer's current and future needs. Without the tool, only one design is going to be produced for the customer because of the time constraint." Our hope is that as more SAN designers use the tool, we will see further evidence of these and other benefits that the HP SAN Designer brings to the SAN design process.

9 Future work

We are fortunate enough to have a steady stream of helpful feedback from the HP San Designer user group, and we are actively pursuing several directions of future work based upon their suggestions. Most importantly, we are adapting our algorithms to accommodate the interoperability constraints between different vendors' devices. We are also working to understand what extensions - if any - are required to apply Appia for the design of Ethernet

SANs and even LANs. A third area of exploration is the design of solutions that provide "slack," to allow graceful growth as requirements evolve. Finally, we plan to extend our algorithms to apply to the design of geographically distributed SANs.

10 Conclusions

The HP SAN Designer, using Appia algorithms, produces high quality SAN designs. Those designs are quite close to the optimal ones, in cases where we can evaluate them directly, and are several times less expensive than some manual designs we have seen, where over-provisioning by a factor of three "just to be safe" is a common approach.

While having cost-effective designs are desirable, it is even more critical to SAN designers that Appia designs can be shown to be correct. The value of reducing human error is extremely high in the complex, mission- and business-critical environments for which SAN design is done.

In summary, we feel that Appia and the HP SAN Designer solve a key, hard problem in storage systems -- and one that is only going to grow in importance as the number, scale, and complexity of the SAN-based storage solutions grows.

10.1 Acknowledgements

For their tireless efforts in making Appia available to the HP storage field, we are indebted to the entire N3SO HP San Designer team: Moises Medina, Lisa Valentini, Günter Hornung, Pascal Melix, Annette Maier, Matthias Binder, Ralf Czepluch, Ergin Er, Sebastian Zangaro, David Russon, Tony Clayton, and Chuck Jaffee. We also benefited from the early enthusiasm of Jerry Duggan, Terry Jackson, and Marie-Jo Fremont.

References

[1] A. Atamtürk, *On capacitated network design cut-set polyhedra*, Research report, IEOR Department, University of California at Berkeley, December 2000, available at http://ieor.berkeley.edu/~atamturk.

[2] D. Bienstock, S. Chopra and O. Gunluk, *Minimum cost capacity installation for multicommodity network flows*, Mathematical Programming **81** (1998), no. 2-1, 177-199

[3] D. Bienstock, and O. Gunluk, *Capacitated network design. Polyhedral structure and computation*, INFORMS Journal on Computing **8** (1996), 243-259.

[4] *Designing next-generation fabrics with Brocade switches*, White paper, Brocade Networks, San Jose, CA, October 2001, http://www.brocade.com/san/pdf/CoreEdgeRev10901.pdf, accessed February 2003.

[5] S. Chopra, I. Gilboa, and S. T. Sastry, *Source sink flows with capacity installation in batches*, Discrete Applied Mathematics **85** (1998), 165-192.

[6] *Overview of Fibre Channel technology*, Fibre Channel Industry Association, http://www.fibrechannel.org/technology, accessed February 2003.

[7] J.W. Herrmann, G. Ioannou, I. Minis, and J.M. Proth, *A dual ascent approach to the fixed-charge capacitated network design problem*, European Journal of Operational Research **95** (1996), 476-490.

[8] K. Holmberg, and Di Yuan, *A Lagrangean heuristic based branch-and-bound method for the capacitated network design problem*, Proceedings of International Symposium on Operations Research, September 1996, pp. 78-83.

[9] T. L. Magnanti, P. Mirchandani, and R. Vachani, *Modeling and solving the capacitated network loading problem*, Operations Research **43** (1995), 142-157.

[10] Li-Shiuan Peh, *The Appia topology solver: implementation*, Technical report HPL-SSP-98-13, Hewlett-Packard Laboratories, Palo Alto, CA, September 1998, http://www.hpl.hp.com/SSP/papers/#Appia.

[11] J. Ward, M. O'Sullivan, T. Shahoumian and J. Wilkes, *Appia: automatic storage area network design*, File and Storage Technologies (FAST) Conference Proceedings (Monterey, CA), January 2002, pp 175-188.