Towards SLA-Based Optimal Workload Distribution in SANs

Eray Gençay*§, Carsten Sinz* and Wolfgang Küchlin*
* WSI for Computer Science, University of Tübingen, D-72076 Tübingen, Germany,
Email: see http://www-sr.informatik.uni-tuebingen.de/pages/staff.html
§ IBM Deutschland GmbH, D-55131 Mainz, Germany, Email: egencay@de.ibm.com

***Abstract*—Storage Area Networks (SANs) connect storage devices to servers over fast network interconnects. We consider the problem of optimal SAN configuration with the goal of meeting service level agreements (SLAs) for server processes while retaining flexibility for future changes. Our approach proceeds in two stages, by setting up Pseudo-Boolean constraint problems and solving them with an off-the-shelf solver.**

First, we give an algorithm for assigning storage devices to applications running on the SAN's hosts. This algorithm tries to balance the workload as evenly as possible over all storage devices. Our second algorithm takes these assignments and computes the interconnections (data paths) that are necessary to achieve the desired configuration while respecting redundancy (safety)requirements in the SLAs. Again, this algorithm tries to balance the workload of all connections and devices in proportion to their capacity. Thus, our network configurations respect all SLAs and provide flexibility for future changes by avoiding bottlenecks on storage devices or switches.

I. Introduction

Today's mainframe computers or servers do not contain disks for mass storage. Instead, business critical data is centrally stored in dedicated large capacity storage devices, which are connected to the host computers (servers) over fast interconnects like fibre channel switches and hubs. Such a Storage Area Network (SAN) may consist of dozens or even hundreds of hosts, switches, and storage devices, and therefore the proper design and management of a SAN is a non-trivial, but business critical, task.

In addition, SAN configurations have to fulfill Service Level Agreements (SLAs), which express performance requirements the SAN has to attain. For example, an SLA may guarantee certain throughput rates or storage capacities for a host application. Many SLAs reflect data flow requirements that originate from the applications running on the servers. While designing a SAN, these requirements should be considered besides "best-practices" constraints like redundancy rules and technical constraints like the limits of the resources, e. g. the number of ports of a device.

Our rule-based system SANchk [1] already checks whether a given SAN configuration respects a number of best practices rules. In this paper, we consider the problem of configuring a SAN in an optimal way, while additionally taking a number of SLAs into account. Our primary concern is not to minimize hardware cost but to maximize the flexibility to accommodate changing SLA requirements in the future. This is because the most critical cost factor associated with a SAN is not raw hardware but operation downtime, e.g. for reconfiguration. In fact, downtime of business critical SANs is simply not an option beyond maybe one hour of scheduled maintenance once a year. Whenever a new or changed SLA cannot be accommodated because reconfiguration is too costly, it must be accommodated by purchasing additional new hardware. Therefore it is of prime importance for a new configuration to avoid future performance bottlenecks which may necessitate major reconfigurations.

Algorithms for the SAN design problem have been mostly developed to minimize the provisioning cost while meeting the QoS requirements and other constraints. In contrast, we attempt to achieve a flexible design by the optimization goal of the most evenly balanced distribution of workloads at all devices. To be more precise, for a given set of hardware devices, we attempt to achieve a uniformly high proportion of free resources at each device, grouped by device types (e.g., X% free storage capacity on each storage device and Y% unused ports in each switch).

In this way, we can increase the probability that the QoS requirements of all applications are still met, even if some of the applications demand more resources than planned, because bottlenecks are avoided. This in turn will result in a decrease of SLA violations and SLA penalties, respectively.

II. SAN Storage Assignment Problem

In the storage assignment problem, we have as input the applications with their requirements (throughput and storage space), the information on which host is serving which applications, and storage devices with their capacities (throughput and storage space). We want to find out, which applications should use which storage devices in order to get a distribution of the workloads on the devices as evenly as possible.

In order to formulate this problem, we use an algorithm that generates all the constraints that are needed, as well as the optimization goal function.

A. Assignment of Applications to Storage Devices

Let $D = \{d_1, \ldots, d_n\}$ be the set of storage devices, and $H = \{h_1, \ldots, h_m\}$ be the set of hosts. Storage devices $d_i \in D$ shall be pairs $d_i = (c_i, t_i)$, where c_i stands for the provided storage capacity and t_i for the provided throughput capacity of device i. In analogy, for all hosts $h_i \in H$, h_i shall also be a pair $h_i = (c'_i, t'_i)$ consisting of the need for storage capacity

c'_i of host i and its throughput requirement t'_i. To express that there is an assignment between a host and a storage device, we define a set $X = \{x_{i,j} \in \{0,1\} \mid 1 \leq i \leq m, 1 \leq j \leq n\}$ with $x_{i,j} = 1$ iff there is a host i that is assigned to storage device j. Obviously, $|X| = |H \times D| = m \cdot n$.

1) Applications and hosts: Let $A_i = \{a_1, \ldots, a_k\}$ be the set of applications that run on host i. Applications a are pairs (c', t') consisting of the storage capacity requirement and throughput requirement. In our formalization, instead of mapping applications to storage devices, we use a simplifying trick: We represent each of the applications on a host as a "pseudo host". Thus, if there are k applications running on host i, we replace the host by k pseudo hosts. By doing this, we just have to map (pseudo) hosts to storage devices. In order to make this simplification step work, we have to conduct an additional trivial pre-check to see if the host can serve all the applications running on it with the desired throughput capacity. Our optimization then produces an assignment of pseudo hosts to storage devices. Since we know which application is running on which host, we can find out in a next step, which physical links are needed between hosts and storage devices.

In our approach, we define an application as an atomic entity that occupies an indivisible block on the storage device. Hence, we would represent a DBMS that uses different storage spaces for its table data and logging information—having different requirements for each of them—as two different applications in our model.

B. Constraint Blocks

In the following, we formulate some Pseudo-Boolean constraint blocks to ensure that we obtain valid assignments to the variables in the set X. We have to encode the following constraints, in which we use $I_H = \{1, \ldots, m\}$ and $I_D = \{1, \ldots, n\}$ as index sets for hosts and storage devices, respectively.

1) Exactly one connection to a storage device for each pseudo host: Every pseudo host (application) should be served by exactly one storage device at the end of the computation. We define this constraint block as follows: $\forall i \in I_H : \sum_{k=1}^{n} x_{i,k} = 1$,

2) Capacity constraints: It has to be ensured that the storage capacities of the storage devices are not exceeded. $\forall j \in I_D : \sum_{k=1}^{m} c'_k x_{k,j} \leq c_j$, i.e., the sum of the storage space requirements of all hosts $h_i \in H$ that are assigned to a storage device d_j ($x_{i,j} = 1$) does not exceed the storage capacity c_j of the device d_j.

3) Throughput constraints: This constraint block should ensure that the port speeds provided by the storage devices are not exceeded: $\forall j \in I_D : \sum_{k=1}^{m} t'_k x_{k,j} \leq t_j$.

C. Optimization Problem

So far, we can find valid assignments between hosts and storage devices. The next step is the definition of a goal function for the optimization. We want the workloads for storage devices to be balanced as much as possible, so that the free resources on devices are maximized proportional to their capacities. This kind of optimization has advantages like the increase of flexibility in the choise of resources, since fewer devices would be working with their full capacity. Another advantage is that the probability of violating service level agreements would also be decreased. Since we maximize the unused part of the devices' resources, it is less likely that the SLAs are violated, even if some of the applications behave unexpectedly.

1) Scaling of inequalities: In order to achieve an equal balancing of the throughputs, we first have to scale all throughput constraints such that their right hand sides become equal. By doing this, we can generate constraints that limit the throughput for each storage device to a constant factor below what is maximally possible. Scaling also serves to obtain integer coefficients, in case the throughput rates are fractions of integers.

Using scaling factors s^*_j ($1 \leq j \leq n$) for our throughput constraints, we obtain: $\forall j \in I_D : s^*_j \cdot \sum_{k=1}^{m} t'_k x_{k,j} \leq s^*_j \cdot t_j$. We define the scaling factors by $s^*_j = 1/t_j \cdot \prod_{i=1}^{n} t_i$. Thus, all right hand sides of the throughput inequalities take the same value $t^*_{\text{norm}} = \prod_{i=1}^{n} t_i$. We might need an additional scaling factor s_I (now the same for all inequalities) to convert the coefficients to integers, but we will not consider this in our further discussion, and assume that after scaling with the factors s^*_j all coefficients are integers.

Obviously, one would like to make the number t^*_{norm} as small as possible in order to reduce the time the solver needs to process the problem. To achieve this, we use a well-known technique, and divide all inequalities by the greatest common divisor (GCD) of all occurring coefficients.

This divisor q is thus computed as the GCD of the set $Q = \{s^*_j \cdot t'_k \mid 1 \leq k \leq m, 1 \leq j \leq n\} \cup \{t^*_{\text{norm}}\}$. Having computed the divisor q, we divide all throughput constraints by this number. We call the resulting scaling factors s_j, i.e. $s_j = s^*_j/q$. Similarly, we call the resulting common right hand side of the inequalities t_{norm}, i.e. $t_{\text{norm}} = t^*_{\text{norm}}/q$.

2) Auxiliary variables: To transform the satisfiability problem we have obtained so far into an optimization problem, we introduce a set of auxiliary variables $L = \{l_0, \ldots, l_p\}$. The auxiliary variables are the binary representation of a number $l = l_0 + 2 \cdot l_1 + \cdots + 2^p \cdot l_p$. These auxiliary variables are added to the scaled throughput capacity inequalities like this: $\forall j \in I_D : s_j \cdot \sum_{k=1}^{m} t'_k x_{k,j} + \sum_{r=0}^{p} 2^r l_r \leq t_{\text{norm}}$.

The auxiliary variables in an inequality symbolize the unused resources on the device. The idea now is that we maximize the value of l.

Since $l_i \in \{0,1\}$ for all auxiliary variables $l_i \in L$, we still have a Pseudo-Boolean problem. The size of the set L, i.e. how large the index p has to be chosen, depends on the right hand side of the inequalities: we have to make sure, that l can cover the whole range from 0 up to t_{norm}. From the fact that the numbers l_i are the binary representation of l, we obtain that $p = \lceil \log_2(t_{\text{norm}} + 1) \rceil$.

3) Objective function: The objective function of the optimization problem maximizes the sum of the auxiliary variables, formally: $\max \sum_{r=0}^{p} 2^r l_r$.

It should be remarked that the values on the right hand sides of the constraints still get potentially very big if large prime numbers are met among the coefficients of the values on the right hand sides. The number of the necessary auxiliary variables depends logarithmically on the size of the normalized right hand side. Thus, a big value on the right hand side of the constraint system increases the solving time very quickly. To achieve better performance at the cost of having a suboptimal solution instead of the optimal one, some big prime numbers in the coefficients of the value on the right hand side can be slightly increased or decreased to solve them in smaller prime numbers. Another way would be to round the right hand sides considering the necessary precision of the parameters.

D. Test of Preconditions

Before generating the constraint system, it is helpful to check some trivial preconditions, whose unsatisfiability implies the unsatifiability of the whole constraint system due to an unvalid configuration of hosts, applications or storage devices. Checking these preconditions, invalid inputs can be avoided and with them time expensive building and solving of a constraint system.

1) Throughput requirements of the applications (i.e. pseudo hosts) are too high: If the sum of the throughput requirements of the applications that are resident on a host is greater than the throughput rate that is supplied by the host bus adapter of that host, then it is impossible to find a satisfying solution. Formally: Let $T_{A_i} = \{t_1, \ldots, t_s\}$ be the set of the throughput requirements of the applications on host i, $t_{\mathrm{sum},i}$ the sum of the throughput rates of the ports of the host bus adapters in host i. If we have $\sum_{t_i \in T_{A_i}} t_i > t_{\mathrm{sum},i}$ the configuration is invalid.

2) Storage space requirements of applications (i.e. pseudo hosts) are too high: If the storage space requirement of a single host or a single application is greater than the greatest available capacity on the side of the storage devices, the configuration is unsatisfiable. Formally: Let C_{A_i} be in analogy to 1) the set of storage space requirements of the applications that are resident on host i, and $c_{\max}$ the maximal storage capacity among the storage devices. Then the configuration is unsatisfiable, if we have $\exists c_k \in C_{A_i}, i \in I_H : c_k > c_{\max}$.

III. SAN Connection Problem

The second SAN configuration problem we consider deals with the interlinks (paths) in a SAN. These interlinks connect hosts, switches, and storage devices, and certain criteria (SLAs) have to be met in order to obtain a correct configuration. For example, it may be required that all paths from hosts to storage devices are laid out redundantly.

We assume that the assignment from hosts to storage devices is already given (e.g., computed by the algorithm we have given in Section II), i.e. we already know which paths are needed and their required throughput.

We assume sets of hosts $H = \{h_1, \ldots, h_k\}$, switches $S = \{s_1, \ldots, s_l\}$ and storage devices $D = \{d_1, \ldots, d_m\}$. By $\mathcal{D} = H \cup S \cup D$ we denote the set of all SAN devices. Moreover, we assume a set of paths $P = \{p_1, \ldots, p_n\}$ to be given which are to be routed through the SAN. Each path $p \in P$ connects a host $h(p) \in H$ to a storage device $d(p) \in D$, but the individual links are not known. Redundancy is modeled by having the same host and storage device connected by more than one path. By $\mathrm{thr}(p)$ we denote the required throughput of a path $p \in P$. Similarly, $\mathrm{thr}(x)$ denotes the maximal throughput for a single port of device $x \in \mathcal{D}$ (we assume the throughput to be the same for all ports of a device). By $\mathrm{ports}(x)$ we denote the number of available ports on device x. To characterize a path, we use predicates $\mathrm{conn}(x, y, p)$, denoting that device x is (directly) connected to device y on path p, and $\mathrm{mult}(x, y, p)$ to denote the required multiplicity of the link between x and y on path p, if it is realized[1]. Note that $\mathrm{mult}(x, y, p)$ can be computed in advance by $\mathrm{mult}(x, y, p) = \lceil \mathrm{thr}(p) / \min\{\mathrm{thr}(x), \mathrm{thr}(y)\} \rceil$. We will also use auxiliary predicates $x \in p$ to denote that device x occurs on path p. Note that $\mathrm{conn}(x, y, p)$ implies $x \in p \wedge y \in p$.

Now we can specify the constraints for a correctly configured SAN network:

1) Each $p \in P$ must be a valid path in the network, connecting device $h(p)$ with device $d(p)$, i.e. there must be a sequence $s_1, \ldots, s_t$ of switches, such that the predicates $\mathrm{conn}(h(p), s_1, p)$, $\mathrm{conn}(s_t, d(p), p)$, and $\mathrm{conn}(s_i, s_{i+1}, p)$ for all $1 \leq i < t$ hold.
2) Redundant paths must not use the same switches. I.e., if paths p_1 and p_2 are redundant—which we will denote by $\mathrm{red}(p_1, p_2)$—then $s \in p_1$ implies $s \notin p_2$ for all $s \in S$.
3) For each path p, the throughput requirement must be satisfied, i.e. $\mathrm{mult}(x, y, p) \cdot \min\{\mathrm{thr}(x), \mathrm{thr}(y)\} \geq \mathrm{thr}(p)$ must hold for all x, y, for which $\mathrm{conn}(x, y, p)$ is true. This constraint always holds if we use the definition for $\mathrm{mult}(x, y, p)$ as given above.
4) The number of ports of each device must be sufficient: $\sum_{p \in P, y \in \mathrm{out}(x), z \in \mathrm{in}(x)} \mathrm{mult}(x, y, p) + \mathrm{mult}(z, x, p) \leq \mathrm{ports}(x)$ for all $x \in \mathcal{D}$, where $\mathrm{out}(x) = \{u \in \mathcal{D} \mid \mathrm{conn}(x, u, p)\}$ and $\mathrm{in}(x) = \{u \in \mathcal{D} \mid \mathrm{conn}(u, x, p)\}$.

As our optimization goal we have chosen to minimize the fraction of ports that are used on each device. This goal ascertains that the load on all devices is equally balanced. It can be expressed as $\min \max_{x \in \mathcal{D}} \left\{ \frac{\mathrm{ports\_used}(x)}{\mathrm{ports}(x)} \right\}$, where $\mathrm{ports\_used}(x)$ is the expression on the left hand side of the inequality in constraint 4. To convert this expression to a Pseudo-Boolean optimization goal, we use the same scaling trick that we already used in Section II-C, i.e. we scale the inequalities in constraint 4 such that they possess a common right hand side, and then introduce a slack variable $l = \sum_{i=0}^{p} 2^i l_i$ on the left hand side of each inequality. Afterwards we maximize the slack variable. We want to conclude this section with two remarks: First, the maximal path length t (in number of switches) is typically quite low, e.g. 3, which facilitates the logical encoding of paths. And second, SAN devices often put a limit on the maximal trunk width (i.e. the number of

[1] We use the predicates $\mathrm{conn}(x, y, p)$ always in such a way that x is closer to the "host side" and y is closer to the "storage device side".

"parallel" links between two devices). This may be specified by an additional restriction similar to constraint 4. All these constraints can be converted to Pseudo-Boolean logic and handled by a standard Pseudo-Boolean solver. Due to space limitations we do not give the details of the conversion here.

IV. Related Work

Optimization problems related to SAN design have been treated previously in several publications ([2], [3], [4], [5], [6], [7]). They differ from our approach mainly in the definition of the optimization problem and its goal function, in the input parameters or methods used to solve the problem.

Ward et al. [2] present two different heuristic algorithms, namely FlowMerge and QuickBuilder, to design a SAN automatically. FlowMerge merges single flows of data that share a switch or hub in a set of flows in a recursive way. Beginning with a fully bipartite, directed graph (every host is connected with every storage device), the number of the edges are decreased incrementally while hub and switch nodes are added to the network. Dicke et al. ([3], [4]) use biologically inspired approaches to improve SAN designs or to create new SAN designs. Walker et al. [8] have a mixed-integer approach to the SAN design problem and also use the minimal provisioning cost as the optimization goal. They focus on the Core-Edge reference topology and provide two formulations for the SAN design problem. A framework for automated storage management based on QoS specifications is Rome [9] by HP Laboratories. Singh et al. [10] have proposed a SAN FS planning tool that uses the notions of application templates and a planning engine to assist a system designer with the design process. PulsatingStore [11] is an analytical framework that provides an automated storage management service for DBMS that balances the conflicting goals of performance guarantees and on-demand resource usage. Anderson et al. [12] present the Disk Array Designer (DAD) that uses a generalized best-fit bin packing heuristic to design disk arrays according to both capacity and I/O performance demands for the application data. Reiss and Kanungo [13] examine the problem of choosing a QoS level for each table or index in a service provider's backend databases to minimize the cost of provisioning storage while satisfying the application level SLAs.

V. Conclusion and Future Work

We defined and formalized the SAN design problem to increase the flexibility of a SAN instead of to minimize the provisioning cost. The flexibility increases on the one hand the ability of the network to grow without the need of major structural changes, and on the other hand its ability to meet the QoS requirements for the applications running on its hosts, even if some of the workloads behave themselves spontaneously irregularly.

The optimization model described in this paper is not exhaustive. It can and should be expanded by further QoS attributes and by more constraints of practical relevance to find solutions that better correspond to the real world problem. The problem model can be enhanced this way. Another issue that can be the subject of future work is enhancing the performance of the algorithms. As we stated before, better performance can be achieved at the cost of having a suboptimal solution instead of the optimal one, by slightly increasing or decreasing the value on the right hand side of the inequality system to solve them in smaller prime numbers. Another way would be to round the right hand sides considering the necessary precision of the parameters. We also want to test our algorithms with large real world examples.

Acknowledgment

We would like to thank Andreas Kübler for his help in implementing the formalization of the assignment problem. The research work for this paper was possible through the funding by IBM Deutschland GmbH. We are grateful to Thorsten Schäfer at IBM Deutschland GmbH, Mainz for his technical support in storage related subjects.

References

[1] E. Gençay, W. Küchlin, and T. Schäfer, "SANchk: An SQL-based validation system for SAN configuration," in *Integrated Network Management*, 2007, pp. 333–342.

[2] J. Ward, M. O'Sullivan, T. Shahoumian, and J. Wilkes, "Appia: Automatic storage area network fabric design," in *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2002, p. 15.

[3] E. Dicke, A. Byde, P. J. Layzell, and D. Cliff, "Using a genetic algorithm to design and improve storage area network architectures." in *GECCO (1)*, 2004, pp. 1066–1077.

[4] E. Dicke, A. Byde, D. Cliff, and P. J. Layzell, "An ant inspired technique for storage area network design," in *BioADIT*, 2004, pp. 364–379.

[5] S. Uttamchandani, G. A. Alvarez, and G. Agha, "DecisionQoS: An adaptive, self-evolving QoS arbitration module for storage systems," in *POLICY*, 2004, pp. 67–76.

[6] S. Uttamchandani, L. Yin, G. A. Alvarez, J. Palmer, and G. Agha, "CHAMELEON: a self-evolving, fully-adaptive resource arbitrator for storage systems," in *ATEC'05: Proceedings of the USENIX Annual Technical Conference 2005 on USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2005, pp. 6–6.

[7] S. Uttamchandani, K. Voruganti, S. Srinivasan, J. Palmer, and D. Pease, "Polus: Growing storage QoS management beyond a "4-year old kid"," in *FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2004, pp. 31–44.

[8] C. Walker, M. O'Sullivan, and T. Thompson, "A mixed-integer approach to core-edge design of storage area networks," *Comput. Oper. Res.*, vol. 34, no. 10, pp. 2976–3000, 2007.

[9] J. Wilkes, "Traveling to Rome: QoS specifications for automated storage system management," in *IWQoS '01: Proceedings of the 9th International Workshop on Quality of Service*. London, UK: Springer-Verlag, 2001, pp. 75–91.

[10] A. Singh, K. Voruganti, S. Gopisetty, A. Fleshler, R. Routray, and C. hao Tan, "SANFS Maestro: Resource planning for enterprise storage area network (SAN) file systems," in *CSREA EEE*, 2005, pp. 32–38.

[11] L. Qiao, D. Agrawal, A. E. Abbadi, and B. R. Iyer, "PULSATING-STORE: An analytic framework for automated storage management," in *ICDEW '05: Proceedings of the 21st International Conference on Data Engineering Workshops*. Washington, DC, USA: IEEE Computer Society, 2005, p. 1213.

[12] E. Anderson, S. Spence, R. Swaminathan, M. Kallahalla, and Q. Wang, "Quickly finding near-optimal storage designs," *ACM Trans. Comput. Syst.*, vol. 23, no. 4, pp. 337–374, 2005.

[13] F. R. Reiss and T. Kanungo, "Satisfying database service level agreements while minimizing cost through storage QoS," in *SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 13–21.