

Conductor: Support for Autonomous Configuration of Storage Systems

Zsolt Németh^a

zsnemeth@sztaki.hu

*MTA SZTAKI Computer and Automation Research Institute
P.O. Box 63, Budapest, H-1518, Hungary*

Michail D. Flouris^b, Renaud Lachaize and Angelos Bilas^c

{flouris,rlachaiz,bilas}@ics.forth.gr

*Institute of Computer Science (ICS),
Foundation for Research and Technology - Hellas
P.O.Box 1385, Heraklion, GR 71110, Greece*

^aWork performed while at FORTH-ICS, P.O. Box 1385, Heraklion, GR 71110, Greece.

^bAlso, with the Dept. of Computer Science, University of Toronto, Toronto, Ontario M5S 3G4, Canada

^cAlso, with the Dept. of Computer Science, Univ. of Crete, P.O. Box 2208, Heraklion, GR 71409, Greece.



CoreGRID Technical Report
Number TR-0057
December 5, 2006

Institute on Knowledge and Data Management

CoreGRID - Network of Excellence
URL: <http://www.coregrid.net>

Conductor: Support for Autonomous Configuration of Storage Systems

Zsolt Németh

zsnemeth@sztaki.hu

MTA SZTAKI Computer and Automation Research Institute
P.O. Box 63, Budapest, H-1518, Hungary

Michail D. Flouris, Renaud Lachaize and Angelos Bilas

{flouris,rlachaiz,bilas}@ics.forth.gr

Institute of Computer Science (ICS),

Foundation for Research and Technology - Hellas

P.O.Box 1385, Heraklion, GR 71110, Greece

CoreGRID TR-0057

December 5, 2006

Abstract

Scalable storage systems are expected to scale to large numbers of physical storage devices and to service diverse applications without incurring high management costs. New storage virtualization architectures and techniques that are currently being proposed, aim at addressing these needs by providing the ability to configure storage systems to meet resource constraints and application requirements. However, this flexibility leads to a large number of options when configuring storage systems either statically or dynamically.

In this work we examine how this process can be automated. We present *Conductor*, a rule-based production system that is able to evaluate alternatives and minimize system cost, based on certain criteria. *Conductor* starts from a set of system resources and a set of application requirements and proposes specific system configurations that meet application requirements while minimizing resource costs. It captures human expertise in the form of rules to generate and evaluate configuration alternatives. In this work we focus on static configuration issues and examine various approaches for reducing complexity within a large configuration space. Our techniques manage to satisfy practical time and resource constraints.

1 Introduction

As the amount of storage required increases, scalable storage systems provide a means of consolidating all storage in a single system and increasing storage efficiency (Figure 1). For this reason, storage system architectures are undergoing a transition from directly- to network-attached. This new architecture offers potential for flexible configuration of storage systems to better match application needs and thus, reduce system cost and improve efficiency. This is an important concern because distinct application domains have very diverse storage requirements; Systems designed for the needs of scientific computations, data mining, e-mail serving, e-commerce, search engines, operating system (OS) image serving or data archival impose different tradeoffs in terms of dimensions such as speed, reliability, capacity, high-availability, security, data sharing, and consistency.

Thus, storage consolidation leads to increased requirements for “flexibility” that will be able to serve multiple applications and their diverse needs. This “flexibility” refers to both storage management and application access issues

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

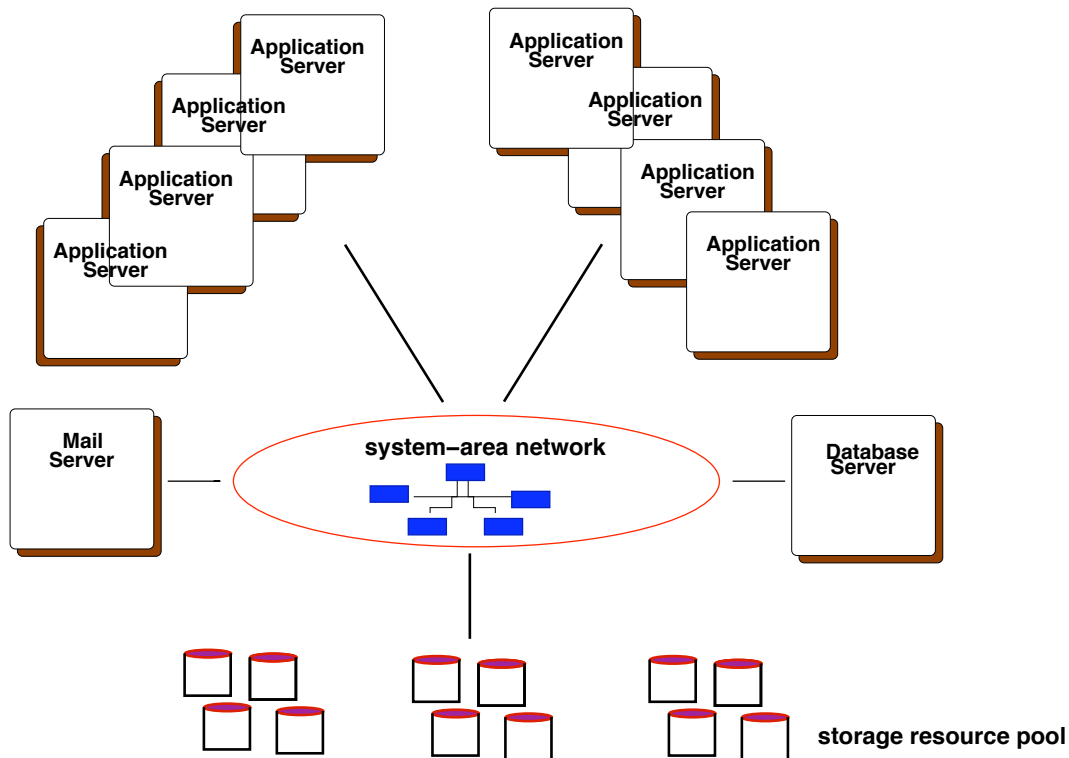


Figure 1: Generic networked storage organization

and is usually provided through “virtualization” techniques: Administrators and applications see various types of virtual volumes that are mapped to physical devices but offer higher-level semantics through virtualization mechanisms.

Modern storage virtualization techniques aim at providing flexibility in configuring and accessing physical system resources. Storage virtualization may occur either at the filesystem or at the block level. Violin [5] is a kernel-level framework for building and combining virtualization functions at the *block* level. Violin targets commodity storage nodes and replaces the current block-level I/O stack with an improved I/O hierarchy that allows for (i) easy extension of the storage hierarchy with new mechanisms and (ii) flexible combining of these mechanisms to create modular hierarchies with rich semantics.

Figure 2 shows a virtual hierarchy that creates a virtual disk by aggregating three virtual devices: an aggregation of two encrypted disks, a pair of striped disks that is encrypted and an encrypted disk. Scenarios of more advanced virtualization semantics are discussed in [6].

Virtualization mechanisms, such as Violin, provide the means required for creating flexible configurations and exporting an abstract view of the actual storage resources to satisfy application requirements. However, such an approach results in a large number of alternatives that can match the application requirements. For instance, a certain capacity can be provided either with a single large disk or by aggregating many smaller disks; A required level of bandwidth may be reached either by a high-performance disk or by striping several lower speed disks; Encryption can be introduced at several levels of the hierarchy: at the topmost virtual device it could represent a centralized service, whereas at the level of physical disks it can be realized as many parallel encryption services. Evaluating these alternatives usually requires human intervention that results in increased management costs. Beyond a certain complexity however, humans cannot survey and manage virtualization hierarchies.

In this work we present *Conductor* that is able to automatically evaluate configuration alternatives and suggest optimal solutions based on certain criteria. *Conductor* captures human expertise in the form of rules. Based on the rules, it builds various alternative configurations and makes suggestions about optimal ones. Configuration is

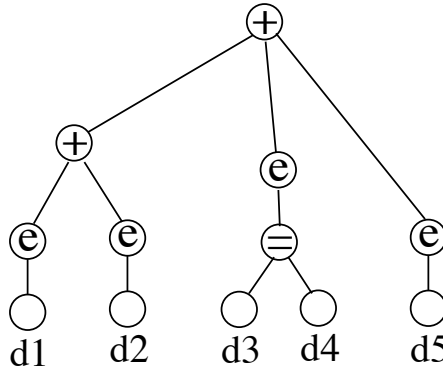


Figure 2: Example virtual hierarchy. ('+' represents an aggregation, '=' a striping, 'e' an encryption virtual device.)

essentially an exploration of a multi-dimensional search space and it is inherently of exponential complexity. The focus of this work is how the complexity of the search can be reduced so that it has acceptable running times for practical cases while the quality of the solution is not decreased.

We find that exhaustive search, i.e. trying all possible configurations is not feasible even for toy examples. Random reduction of the search space is a potential alternative but may exclude some of the optimal solutions. Heuristic search applies some additional information about the search space. We investigate two methods: (a) Zoned search, which enables or prohibits certain actions according to specific zones within the configuration space. (b) Clustered search, which creates clusters of similar devices and uses these clusters as the basis of the configuration space search. Both of these methods are able to deliver optimal solutions, have practically acceptable complexity and, in addition, can be combined.

The rest of this paper is organized as follows. Next section introduces related work. Section 3 presents an overview of *Conductor*. Section 4 discusses quality and complexity issues for various approaches to evaluating alternative system configurations. Section 5 presents our results and analysis of the overheads associated with each approach we examine. Finally, we draw our conclusions in Section 6.

2 Related work

Storage management involves many problems that are hard to formalize, involve multi-dimensional optimizations, exponential search, or ambiguous decisions. Even if there are explicit algorithms for certain problems, they quite often belong to the NP-hard class. Recent work tries to exploit “intelligent”, heuristic-based, approaches for tackling some of these problems.

Polus [11] aims at mapping high level QoS goals to low level storage actions by introducing learning and reasoning capabilities. The system starts with a basic knowledge of a system administrator expressed as “rules of thumb” and it can establish quantitative relationships between actions taken and their observed effects to performance by monitoring and learning. To eliminate performance problems, the system finds an appropriate set of actions by backward reasoning in the generated knowledge base.

Ergastulum [2] is aimed at supporting the configuration of storage systems. It essentially helps with reducing the search complexity of possible design decisions by utilizing a best-fit bin packaging heuristics with randomization and backtracking. It takes into consideration workload characteristics, device specifications, performance models and constraints, and provides a near-optimal solution in practically acceptable time.

The work introduced in [3] assists in selecting the right data-redundancy scheme for disk arrays. It is a derivative of Ergastulum and explores and evaluates four methods for a specific problem: rule-based tagger, model-based tagger, and partially and fully-adaptive solvers.

A novel approach presented in [10] tries to predict the effect of certain actions and helps with making decisions at data distribution (encoding and placement). It has some similarities with the learning abilities of Polus. It establishes a set of *What... if...* statements where the hypothetical effect (what part) of a certain circumstance (if part) is stored. These relations are obtained by statistical, analytical or simulation methods. The accuracy of predictions were shown

to be practically acceptable.

A main effort in both our as well as previous research is to capture aspects of human expertise. However, previous research has focused on different aspects and has applied different techniques. In our work we examine configuration of the virtualization hierarchy and consider disks themselves as black boxes, i.e. exclude low level physical aspects in the investigation. Polus mainly operates at the physical disk level and does not consider structural issues. Ergastulum and its derivative is similar to our work in a sense that it is also aimed at reducing the complexity of configuration. However, it supports initial system designs only and strongly relies on assumptions of the workload and performance models. Our intention is to extend Conductor so that it can manage dynamic reconfigurations therefore, initial configuration is carried out with limited assumptions about the workload and estimated performance figures. Also, the search methods used in each case are different: Ergastulum uses backtracking, whereas we apply forward chaining. At the current phase of work we do not apply any learning or predictive abilities nevertheless, these may be considered in the future.

3 System Overview

The aim of Conductor is to increase the degree of autonomy in storage systems by creating and maintaining virtual storage hierarchies (customized storage services) based on user and application requirements, without the intervention of system administrators. Thus, Conductor partly substitutes human system administrators and overtakes some of their tasks during the life-cycle of a storage system:

1. Initially configure new virtual hierarchies based on a prescribed specification.
2. Monitor hierarchies to ensure they satisfy at runtime the prescribed specification.
3. Detect where problems occur in hierarchies that deviate from specifications.
4. Modify or rebuild, partly or entirely, such hierarchies.

In large-scale storage systems today, such tasks are performed by experienced personnel and thus, rely heavily on human expertise. As storage systems grow in size and their architectures leverage commodity components, it is projected [1] that the cost of maintaining them may dominate and eventually limit their scalability.

In this section we present the overall concept of Conductor. Although Conductor aims at addressing all issues above, this paper focuses on the first step, the creation virtual storage hierarchies based on prescribed specifications. This is the static part of the above tasks. The rest of the tasks constitute dynamic runtime management steps and are left for future work.

3.1 Conductor architecture

Conductor is built in the context of Violin [5], a storage virtualization framework. Our approach is to augment Violin with a rule-based, forward chaining production system for the following reasons: (i) Rule-based systems offer a straightforward way to express “common sense” and ”rules of thumb” that are based on human expertise. These rules are declarative and specify the knowledge not the procedure to solve a problem. (ii) Data are represented as facts, i.e. abstract statements. This uniformity makes it possible to represent virtually all sort of information. (iii) Production systems apply highly efficient pattern matching mechanisms to select appropriate rules. Therefore, they are able to capture information hidden in unrelated, unstructured, heterogeneous data by searching for certain patterns.

Figure 2 depicts the conceptual architecture of Conductor. Conductor communicates with the storage system (Violin) at two well defined points: it receives monitoring data from the storage system and sends configuration commands to it. Monitoring data are transferred between Violin and Conductor via a simple interface, such as the `/proc` filesystem in Linux. Commands that dictate configuration changes to Violin can also be sent through a similar interface, such as `ioctl` calls in Linux. All information is represented in Conductor as facts and rules:

Facts represent factual information about the system, that may be (i) static, which is basic information about system components, e.g. a directory of disks, network links, host nodes and their characteristics; (ii) dynamic, which is measured during system operation, e.g. throughput obtained from each virtual device, number of requests traversing a path; (iii) inferred, which is derived by Conductor during its working cycle.

Rules represent empirical knowledge about the system. They express actions that must be performed in a certain situation [4]. Such situations are described by the conditions of the rule that must be satisfied to enable the actions.

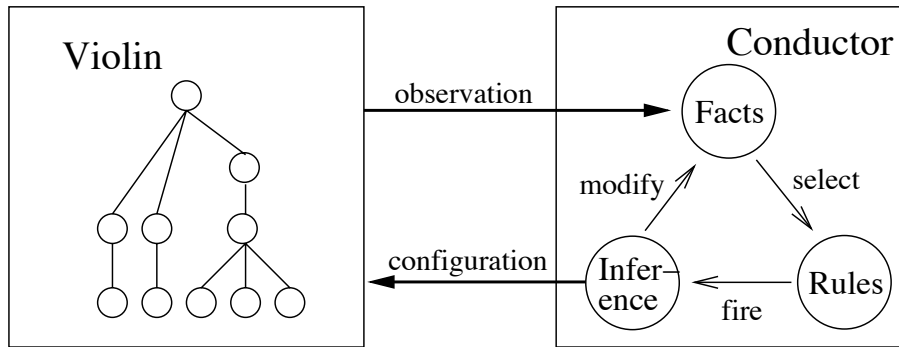


Figure 3: Conductor architecture.

Conditions involve the existence or non-existence of facts or certain patterns that facts must satisfy. From all applicable rules, i.e. where conditions are true, one is chosen and its actions are executed. Rules in Conductor essentially try to capture "rules of thumb" a human operator would perform in each case.

Conductor is implemented in CLIPS [4, 8], a production system framework, that realizes the production cycle [7] as depicted in Figure 2 :

1. Update factual information in the knowledge base. Facts activate rules as they are matched with the conditions of the rule.
2. From the potentially many activated rules one is selected according to the conflict resolution strategy of the production system.
3. As the selected rule fires, it may generate new (inferred) facts and update the knowledge base.

This simple cycle continues as long as there are activated rules. Note, that in each production cycle rules may be activated or deactivated and the execution order is governed by the conflict resolution strategy.

Conductor operates in two possible modes:

- Initial configuration mode. This mode is used to configure virtual devices based on user requests and static facts.
- Diagnosis and dynamic reconfiguration mode, where it triggers diagnostic procedures and corrective actions. This work focuses on the initial configuration mode, which we describe next in detail.

4 Initial Configuration Mode

Initial configuration is a "bootstrapping" process that creates a new virtual hierarchy (target device) of devices from scratch according to a set of user and application requirements. Some of these requirements are related to the functionality of the target storage device and have static characteristics, whereas other requirements capture performance aspects and are dynamic. While static requirements can be guaranteed for the lifetime of the target device, the exact performance of a target device is hard to predict and guarantee. Storage performance is related to workload characteristics, such as access patterns that are typically unknown, difficult to predict, and dynamic in nature.

Rules used in the initial configuration mode satisfy all static requirements, whereas they only "try" to satisfy dynamic (performance) requirements based on estimated values for system components. One possible approach to improve this is to dynamically update estimated values with actual measured data as soon as they become available and trigger a full system reconfiguration, based on these measured values.

Conductor needs to explore a large configuration space with multiple dimensions. Each dimension corresponds to a property of the disks, e.g. capacity, bandwidth, level of redundancy. Some of these properties are continuous, whereas others are discrete. Any device is represented in this space by a vector: Each request for a target device is translated to a vector with components derived either from the user request or from desired system characteristics. Physical disks are represented as vectors with estimated values for their performance characteristics. Virtual devices that are created from combining other (virtual or physical) devices are represented as vectors with components derived

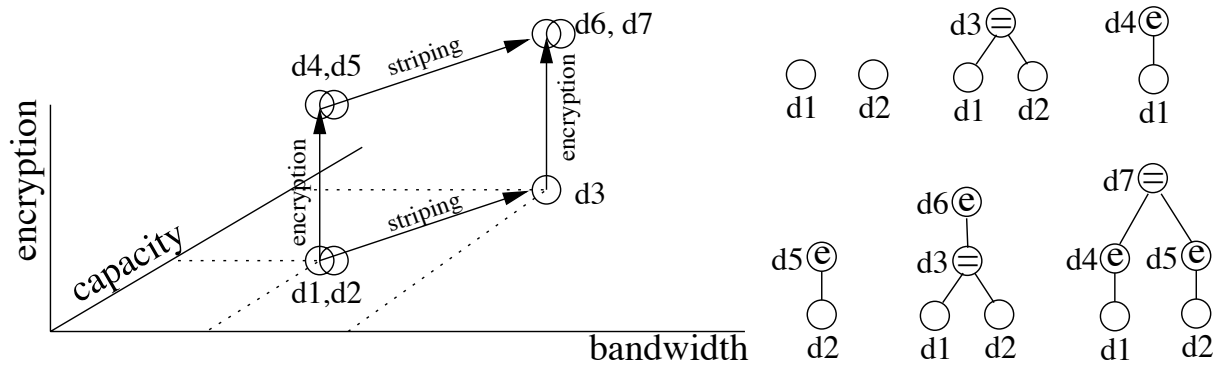


Figure 4: Configuration as exploration of the search space and the generated disks. (Empty circles are physical disks, = represents a striping, e represents an encryption virtual disk.)

by production rules. These rules express relationships and constraints between dimensions of device properties and guide configuration steps. Thus, we can define the problem of configuration as a search procedure in this multi-dimensional space that tries to produce a given target vector, the user request, using only appropriate combinations and modifications (rules) of initial vectors (devices). A solution to the problem consists of the set of initial vectors and the set of rules applied to transform them into the target vector.

Figure 4 shows an example with a configuration space of three dimensions. Initially, there are two disks d_1 and d_2 of a certain capacity and bandwidth and none of them are encrypted. If the rule of striping is activated by d_1 and d_2 and fires, it yields d_3 that has the added capacity of d_1 and d_2 and a bandwidth increased by factor α ; for instance, assuming a linear access, α is around 2. When the rule of encryption is activated by d_3 and fires, it yields d_6 with the same capacity and bandwidth but with the added service of encryption. Rules are activated by disks and combinations of disks in an unspecified order if certain conditions are met. Thus, the rule of encryption can be activated by d_1 and d_2 , yielding d_4 and d_5 that activate the striping rule yielding d_7 with the same characteristics as d_6 . If any of these disks fall into a defined range of the required parameters then the search for a given configuration is successful.

Note however, that this example is simplified. There can be significantly more rules active simultaneously, e.g. aggregation, 3-, 4-, 5-way striping, mirroring, partitioning which combined would generate a huge disk population in the search space. Furthermore, only a few dimensions are taken into consideration in this example. For instance, encryption does not change capacity or bandwidth but it may increase response time. If mapping is also taken into consideration as a further dimension, d_6 and d_7 do not necessarily coincide. d_7 consists of two encrypted virtual disks that may work in parallel – depending on mapping options – giving better performance than the single encryption layer of d_6 .

The search method may generate large numbers of possible disk configurations within an acceptable distance to the requested specification vector but one of them must be chosen, eventually. The specification can have more components than the required parameters. These additional components are related to the management of the device and may involve dollar cost, resource utilization, structural complexity, power consumption or other aspects; some are defined by the user, some others by the service provider. The specification vector may also give weights and annotations how these metrics can be evaluated. Based on these metrics a function is calculated that is a linear combination of certain scores how much the given component fits the specified one. For instance, the resulted capacity should be as close to the specified one as possible whereas the power consumption should be as little as possible furthermore, there are components, like encryption that requires an exact match. With appropriate weights the importance of these aspects can be tuned from don't care to uttermost. The calculated function is used to select the best vector among the resulted ones. Intuitively, we may say that we choose the closest vector to the specified one but the calculated function is not a distance metrics in a strict sense, hence we call it "cost". Smaller cost means better solution. In current experiments we use a simple dollar cost metrics; certain cost functions will be defined based on practical experience.

It is important to note that the search is non-monotonous: Virtual devices closer to the required specification vector are not necessarily better or more useful in generating the final solution. For instance, a given bandwidth can potentially be provided by two disks of 60% of the requested bandwidth whereas the same could not be fulfilled efficiently by two disks that have 90% of the requested bandwidth.

Also, the search is non-exact. Its aim is to find a target device with characteristics as close to the user requirements

as possible. However, in most cases, the resulting device will not be an exact match, especially in cases where dynamic characteristics are considered.

4.1 Rules

Configuration is driven by rules that specify three aspects for creating a new virtual device from existing ones: (a) What properties the existing devices need to have, (b) Specifically how they should be combined, and (c) What are the projected property values for the new device.

There are different rules for various combinations and modifications of disks, e.g. aggregation, (n-way) striping, mirroring, partitioning, encryption. They incorporate conditions (disks eligible as targets of the operation), constraints (under what circumstances the operation is possible) and effects (new or modified parameters) of the given operation. For instance, the rule of striping picks two disks and check if (i) their capacities are the same, (ii) their bandwidths are similar, (iii) their latencies are below a certain limit and (iv) either both of them or none of them are encrypted. If these conditions hold, then striping is possible and results in a new disk that has doubled capacity, increased bandwidth, the same latency and encryption as the constituting disks.

As rules fire, they create new devices in the search space. These may fire other rules that, in turn, create new devices. This could continue infinitely therefore, rules incorporate conditions that limit the search, e.g. the height of the hierarchy or the distance from the requirements. When there are no more activated rules, the solutions are checked. If there are disks within an acceptable distance from the requested specification vector, one of them is chosen based on the cost function. Otherwise, some limiting conditions are changed that activate rules, e.g. one level higher hierarchy is allowed, and the search continues. It stops finally if either a solution is found or there are no solutions in a given number of iterations. To control the complexity of the search process we use certain heuristics. Next these are examined in details.

4.2 Exhaustive search

With exhaustive search all possible vectors are generated according to the rules. Some of these vectors will lead to the requested specification vector, whereas others will not. This approach will eventually generate the target device that is the closest to the requested specification at the least possible cost. However, this method has exponential complexity and is not realistic in practice.

A simple approach to reduce the complexity of the exhaustive search is to randomly omit generated vectors at the search. In our evaluation, we examine dropping one every 16, 8, 4, and 2 generated devices randomly. Although this simple approach significantly reduces the search space, it cannot guarantee that the generated disk will be acceptably close to the requirements and, even in this case, that it is the smallest possible cost.

4.3 Zoned search

Zoned search is heuristic, i.e. uses some additional information in order to omit irrelevant solutions and thus, decrease the search space. As rules define the projected properties, some disks can be combined so that they lead to the required solutions, whereas some others can not; they form certain zones in the search space. Zoned search divides the search space into zones and ignores those that cannot yield the required characteristics. As an example, consider a 2-dimensional search space in Figure 5(a). Assume the following: solutions are accepted with a 10% margin around the required vector (zone A in Figure 5(a)) and the rule of striping assumes an increase of bandwidth of 2. In this case no disks of bandwidth in the 55% – 90% region of the required can lead to a solution. They need additional bandwidth to be provided by striping. However, striping would arrange the new disk over the 110% range of the desired bandwidth. Similarly, disks with capacity above the 55% limit would not lead to the required solution. By striping these disks, the resulted capacity would be over the required. The idea can be followed in a recursive way, e.g. for zone B; in such a way the entire search space can be divided and classified into allowed and prohibited zones. This method can be generalized easily for other actions where the increase of bandwidth is other than 2.

It is important to note that zones are allowed or prohibited with respect to some action. For example, allowed zones for a 3-way striping mostly coincide with the prohibited zones of a 2-way striping. In the above example zones that may lead to capacity or bandwidth over 110% of the required are considered prohibited zones. However, if partitioning is an allowed action, these zones are allowed, since they can reach the target zone by creating a disk with much larger capacity and bandwidth and then use partitions of it.

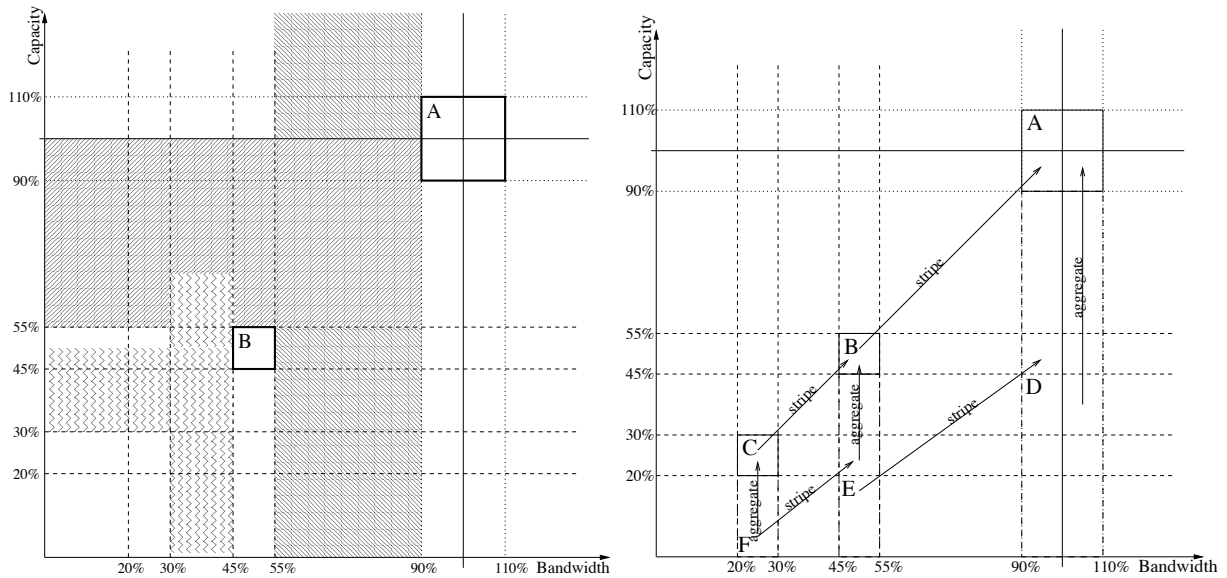


Figure 5: (a) Valid and invalid zones in the search space (left). (b) Operations are enabled one at a time so that zone A can be reached in the fewest steps (right).

4.4 Zoned search with random dropping

Zoned search may reduce search space significantly. However, within the allowed zones it still performs an exhaustive search. In the worst case scenario, when all disks can potentially lead to a solution, zoned search coincides with exhaustive search. Therefore, zoned search can also be combined with random dropping.

4.5 Fewest steps

The method of fewest step is also based on the zones. In this case however, combination actions are enabled one at a time. First, configuration checks if there are disks in the target zone (zone A in Figure 5(b).) If there are no disks then it checks those zones where zone A can be reached in a single step. Therefore, it enables aggregation for zone D and then 2-way striping for zone B. If there are still no solutions, it enables further rules that can lead to zones B and D in a single step: aggregation can put disks from zone E to zone B whereas 2-way striping can put disks from zone E to zone D. Obviously, more way striping can also be considered with different zones. As long as there are no solutions, it checks recursively, how these zones can be reached with the fewest steps. It cuts down large parts of the search space, however, with a high likelihood of not finding the solution with the lowest cost.

4.6 Device clustering

The above search methods may assume a random disk population, i.e. they can be scattered evenly in the parameter space. In practice however, there are many disks of the same type and specifications. A clustering method puts these disks into groups and takes into consideration only one representative element of them. Groups are established dynamically as the configuration process goes on. The method significantly reduces search complexity. For instance, if there are n identical disks, the complexity of aggregation is $O(n^2)$ ($n * (n - 1)/2$ potential pairs), whereas that of in case of clustering is $O(n)$ ($n/2$ pairs.) In general, complexity is proportional to the number of disk groups and not to the number of disks. While there may be hundreds or thousands of disks in a storage system, the number of groups is significantly lower. Note, that this method is orthogonal to the previous ones, i.e. it can be exhaustive but could be combined with zoning or random dropping. Its worst case scenario, when all groups have a single element, has the same complexity as its non-clustered version.

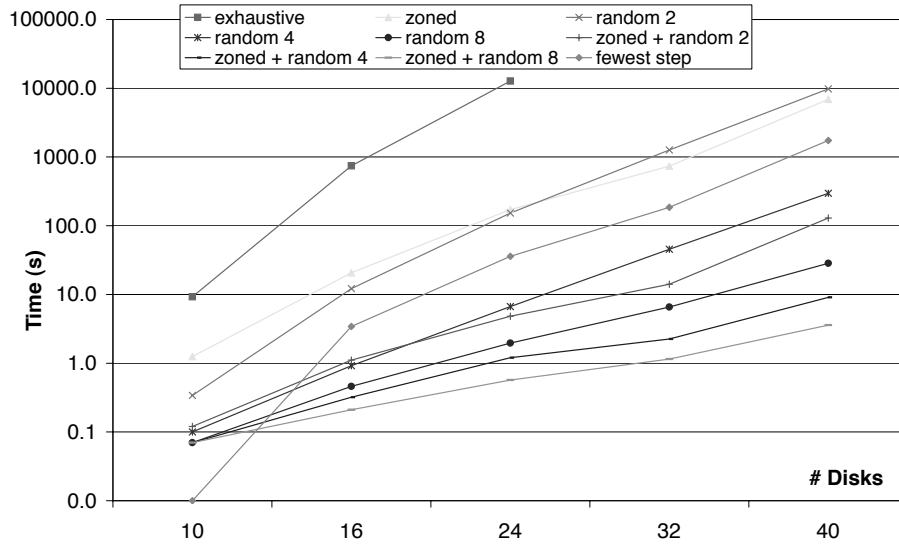


Figure 6: Run times of different configuration search methods.

5 Evaluation

We have implemented the framework of Conductor using CLIPS [4, 8]. Our implementation currently provides rules for aggregation, 2-, 3-, 4-, 5-way striping, encryption and mirroring with all the search heuristics mentioned in the previous section. In this section, we evaluate the search heuristics with respect to run-time complexity. We first present results without disk clustering and then we consider the effect of clustering similar disks in groups. We run our experiments on PCs with Intel P4-grade CPUs at 3.0 GHz and 512 MBytes of memory.

5.1 Individual Disks

Figure 6 summarizes the search times without clustering. The system consists of a variable number of disks (x axis). Since in this experiment disks are taken individually, we assumed a random disk population where each disk has a capacity of 100, 200, 300, 400, and 500 GBytes and nominal bandwidth of 33, 50, 70 and 100 MBytes/s. Also, each has an initial cost, a one-dimensional integer metrics that is used to select the 'cheapest' solution among the functionally equivalent ones. For the sake of simplicity we may consider it as the dollar cost of the resource. A user asks for a virtual disk with 600-GByte capacity, 100-MBytes/s throughput, and a cost not exceeding 200. For the purpose of this experiment, Conductor is allowed to use only two actions: aggregation and striping. All other actions are disabled.

We see that although exhaustive search finds the best solution possible (minimum cost), it is in practice unable to deal with more than 24 physical disks. In this case it generates more than 200000 facts and fires nearly 120000 rules, which results in unacceptably long running times (and even sometimes crashes).

If we omit some of the intermediate virtual disks *randomly* and keep only one out of two, four, and eight (labeled as random 2, random 4, random 8 in Figure 6), the search becomes viable both in compute time and memory capacity. This method, however, eliminates as expected some of the best solutions. Figure 7 compares the cost of the best solutions found. We see that compared to exhaustive search, these methods result in disks with higher cost. However, the difference decreases as the number of disks grows.

Zoned search reduces the search space efficiently while it does not eliminate good solutions: It results in virtual

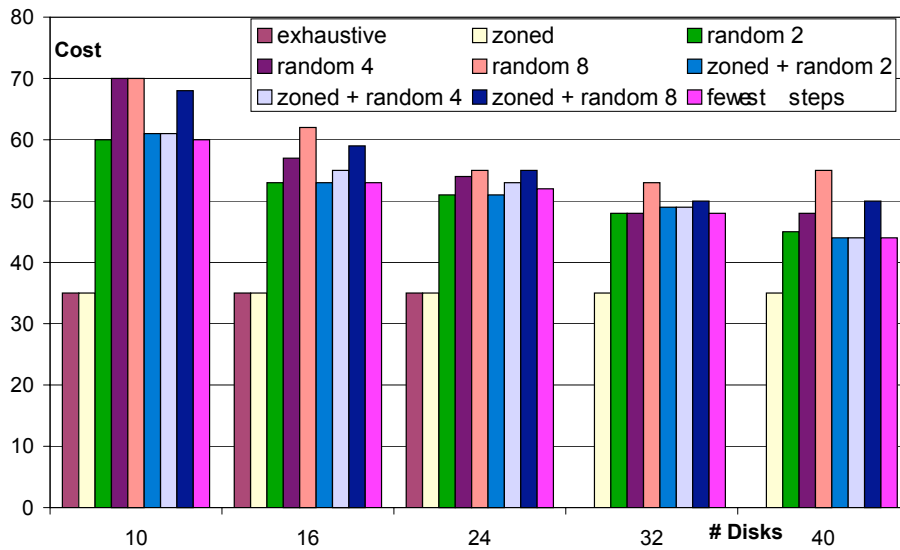


Figure 7: Comparison of the costs of the best solution found by different methods.

disks with cost similar to exhaustive search and can handle significantly more disks. Furthermore, zoned search can be accelerated by applying random dropping and renders running times acceptable even for more than 40 physical disks.

Finally, it seems that fewest-steps is an interesting but not practically useful approach. Its running time is inconsistent: if it finds a solution in a small number of steps (1 or 2), as is the case with 10 physical disks, then it is significantly faster than other methods. Otherwise, its running time is close to zoned search, however, with potentially higher cost in the resulting virtual disk configurations.

5.2 Clustered Disks

In practice, hardware resources, such as disks, are often acquired and upgraded in bulk and exhibit similar characteristics. Thus, we can assume that, in most cases, a large system will consist of disks that can be clustered based on their characteristics to a smaller number of groups. As opposed to the previous experiment, where potentially all disks may be different, in this experiment we introduce 3, 5, and 7 different groups of disks. Since we anticipate that clustering will result in improved running times, we use a more complex example: A user requests a virtual disk with capacity of 1200 GBytes and throughput 140 MBytes/s. Moreover, we configure two versions of this disk, with and without encryption. To increase the size of the configuration space, we enable in Conductor rules associated to aggregation and 2- and 3-way striping. Finally, we only present results for exhaustive search to see the net effect of clustering.

Figure 8 shows that running times are approximately three orders of magnitude smaller compared to using individual disks. Even for 128 disks of 7 different types the search is less than 2 minutes. When including encryption in the target virtual disk configuration, the number of groups that needs to be considered doubles. However, running time remains low.

6 Conclusions and future work

Future, large-scale storage systems are envisioned to offer a lot of flexibility in configuring virtual resources to meet user and application requirements. Configuration of large-scale storage systems that support such flexibility requires

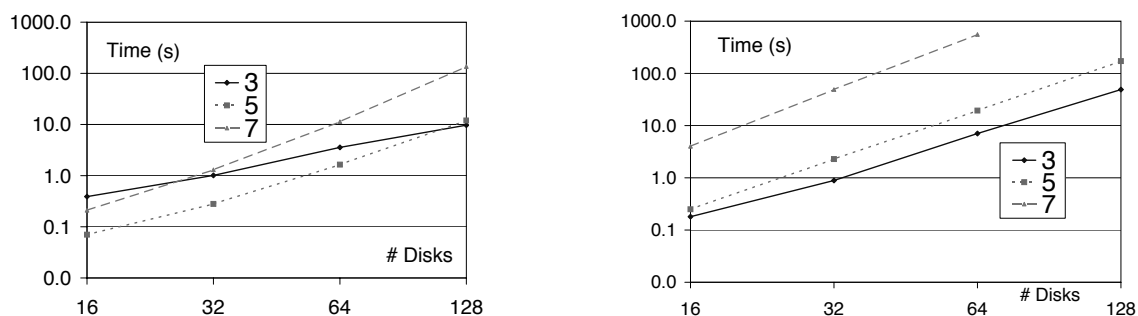


Figure 8: Run times of clustered disk search for 3, 5, and 7 groups of disks, without and with encryption.

considering a large number of alternatives. For instance, a user request for a virtual volume of specific size, throughput, redundancy level, with encryption and compression capabilities over a system that consists of hundreds of physical disks will require considering millions of alternatives. Although this task is traditionally performed manually by experienced storage administrators that can quickly reduce this space, it is foreseen that human cost will not scale with system size.

In this work, we propose Conductor, a rule based system for evaluating configuration alternatives that meet user requirements and minimize system cost. Although Conductor is designed to deal with both static (initial configuration based on estimated performance values) and dynamic (run-time reconfiguration based on measured performance values) properties of storage systems, in this work we only explore initial system configuration. The main issue in this direction is to reduce search complexity.

Our design relies on heuristic rules that capture human expertise and various search methods that aim at reducing search complexity without compromising the quality (cost) of the resulting configurations. We implement Conductor as an extension to Violin [5] using CLIPS [4]. We find that although considering individual disks may be in practice prohibitive for real, large-scale systems, clustering disks in groups substantially improves overheads and results in a practical approach to exploring the configuration space.

The next step in our work is to explore dynamic system behavior, i.e. what is the quality of the proposed configurations at runtime, how far is it from user requirements in terms of dynamic features such as throughput and response time, and how we can incorporate more knowledge in system rules to allow for run-time reconfiguration actions that will result in improved configurations. Finally, besides dynamic configuration issues, future work should also consider mapping of virtual volumes to distributed physical resources and should take into account disk, CPU, memory, and network characteristics.

7 Acknowledgments

During this work Zsolt Nmeth was supported by the ERCIM Fellowship programme. Also, we thankfully acknowledge the support of the European FP6-IST program through the UNiSiX project and the FP6 Network of Excellence CoreGRID, and the support of the General Secretariat for Research and Technology, Greece through the MASC project.

References

- [1] Gartner Group. Total Cost of Storage Ownership – A User-oriented Approach. September 2000.
- [2] E. Anderson, M.Kallahalla, S. Spence, R. Swaminathan, Q.Wang. Ergastulum: Quickly Finding Near-Optimal Storage System Designs. HP Laboratories SSP technical report HPL-SSP-2001-05 (2002)

- [3] E. Anderson, R. Swaminathan, A. C. Veitch, G. A. Alvarez, J. Wilkes. Selecting RAID Levels for Disk Arrays. Proceedings of the USENIX FAST Conference on File and Storage Technologies, January 28-30, 2002, Monterey, CA, USA.
- [4] C Language Integrated Production System (CLIPS) <http://www.ghg.net/clips/CLIPS.html> and also CLIPS Reference Manual. Vol. 1, Version 6.24, 15 June 2006
- [5] M.D. Flouris, A. Bilas. Violin: A Framework for extensible Block-level Storage. Proc. of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2005) April 2005, Monterey, CA, USA. IEEE Computer Society.
- [6] M. Flouris, R. Lachaize, and A. Bilas. Violin: a Framework for Extensible Block-Level Storage. Book on Knowledge and Data Management in Grids, CoreGRID series, Springer Verlag, 3, 2006.
- [7] D.Klahr, P. Langley, R. Neches (eds.): Production System Models of Learning and Development. MIT Press, 1987.
- [8] W. Mettrey. A Comparative Evaluation of Expert System Tools. Computer 24, 2 (Feb. 1991), pp. 19-31.
- [9] E. Riedel. Storage Systems. Not Just a Bunch of Disks Anymore. Queue, June 2003, ACM, pp. 32-41.
- [10] E. Thereska, M. Abd-El-Malek, J. J. Wylie, D. Narayanan, G. R. Ganger. Informed data distribution selection in a self-predicting storage system. Proc. of the International Conference on Autonomic Computing, ICAC-06, Dublin, Ireland, June 2006.
- [11] S. Uttamchandani, K. Voruganti, S. Srinivasan, J. Palmer, D. Pease. Polus: Growing Storage QoS Management Beyond a "Four-year Old Kid". USENIX FAST '04 Conference on File and Storage Technologies, March 2004, San Francisco, CA, USA.