Characterizing Data Dependence Constraints for Dynamic Reliability Using *N*-Queens Attack Domains

Ulya Bayram, Kristin Yvonne Rozier, and Eric W.D. Rozier(✉)

University of Cincinnati, Cincinnati, OH, USA
bayramua@mail.uc.edu, {Kristin.Y.Rozier,Eric.Rozier}@uc.edu

Abstract. As data centers attempt to cope with the exponential growth of data, new techniques for intelligent, software-defined data centers (SDDC) are being developed to confront the scale and pace of changing resources and requirements. For cost-constrained environments, like those increasingly present in scientific research labs, SDDCs also present the possibility to provide better reliability and performability with no additional hardware through the use of dynamic syndrome allocation. To do so the middleware layers of SDDCs must be able to calculate and account for complex dependence relationships to determine an optimal data layout. This challenge is exacerbated by the growth of constraints on the dependence problem when available resources are both large (due to a higher number of syndromes that can be stored) and small (due to the lack of available space for syndrome allocation). We present a quantitative method for characterizing these challenges using an analysis of attack domains for high-dimension variants of the n-queens problem that enables performable solutions via the SMT solver Z3. We demonstrate correctness of our technique, and provide experimental evidence of its efficacy; our implementation is publicly available.

Keywords: Big data · Reliability · Storage · n-queens · Intelligent systems

1 Introduction

One of the largest challenges facing the storage industry is the continued exponential growth of Big Data. The growth of data in the modern world is exceeding the ability of designers and researchers to build appropriate platforms [11,26] but presents a special challenge to scientific labs and non-profit organizations whose budgets have not grown (and often have been cut) as their data needs steeply rise. The NASA Center for Climate Simulation revealed that while their computing needs had increased 300 fold in the last ten years, storage had increased 2,000 fold, and called storage infrastructure one of the largest challenges facing climate scientists [8]. This trend has been driving reliance on commercial off the shelf (COTS) solutions to drive down the cost of data ownership. Despite

J. Campos and B.R. Haverkort (Eds.): QEST 2015, LNCS 9259, pp. 211–227, 2015.
DOI: 10.1007/978-3-319-22264-6_14

its importance, the goal of affordable data curation comes at a cost in terms of reliability creating a difficult to solve system design constraints problem.

To cope with the increase in cost, deduplication techniques are commonly used in many storage systems. Deduplication is a storage efficiency improvement technique that removes the duplicate substrings in a storage system and replaces them with references to the single location storing the duplicate data. While this achieves a higher storage efficiency in terms of reducing the cost of ownership of a system, it can negatively impact the reliability of the underlying storage system since loss of a block with high number of references means a critical number of files being lost unrecoverably [22].

Data reliability was previously improved using enterprise-class storage devices that typically suffer faults as much as two orders of magnitude less often than COTS storage devices. In the face of the exponential growth of the digital universe [28] the cost of this solution has become prohibitively expensive, inspiring a switch to near-line components, thus lessening storage reliability guarantees. While reliability could be improved through the addition of new hardware, today the scale of growth of inexpensive storage is being exceeded by the growth of Big Data.

In most storage systems reliability improvements are achieved through the allocation of additional disks in Redundant Arrays of Independent Disks (RAID) [20]. RAID arrays achieve reliability through the allocation of coding syndromes [21] which create dependence relationships in the storage system to allow recovery of files after failures. While RAID systems are incredibly effective at the task of improving reliability, they add to the cost of the storage systems in which they are deployed.

Methods used to increase reliability also increases the cost of maintaining the storage system, and same is true for the methods that reduce the cost; they also reduce reliability. In order to meet these cost and reliability constraints, and find a way to break the proportional relationship in between, previously we have conducted a study where we have documented that systems are often over-provisioned, and this over-provisioning level is highly predictable using intelligent systems algorithms [23]. Using these models, we have proposed that dynamically allocated reliability syndromes could be created and stored in this excess capacity to improve reliability without the addition of new hardware [3]. Based on this result, it is now possible to modify traditional RAID schemes to dynamically allocate new syndromes for reliability in over-provisioned space through the risk-averse prediction of available storage over the next epoch of operation of a storage system. Furthermore this can be done while maintaining quality of service (QoS) and availability of the storage system while simultaneously providing maximum additional reliability. The only assumption is that the additional syndromes can be placed in a way that respects data dependence constraints. The ability to predict the expected level of over-provisioning allows us to create software defined data centers which can allocate virtual disks made up of free space compiled from across the data center to hold additional reliability syndromes. An unsolved challenge which stands in the way of this technique, however, is the development of algorithms that account for complex data dependencies such as

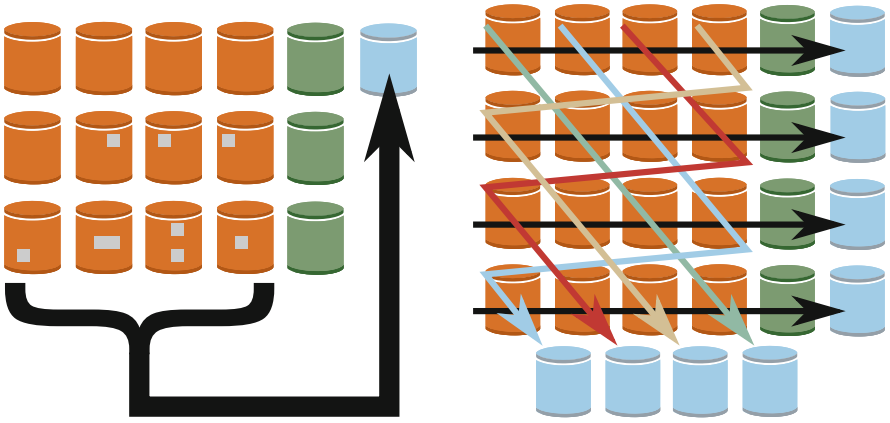
existing reliability syndromes and deduplication, providing a strategy for syndrome storage and new RAID relationships in a performable way that maximizes the additional number of reliability syndromes that can be allocated without violating the dependence constraints on those syndromes.

In order to solve these dependence constraints we cast our problem into a unique variant of the n-Queens problem. We map a RAID array into a mathematical representation of a chess board with a set number of *ranks* (defining the y-axis) and *files* (defining the x-axis). We propose a quantitative solution for virtual disk allocation in software defined data centers, respecting all dependence constraints within the data center, or, when no such configuration exists, identifying the unsatisfiability of the problem. This method allows us to take advantage of the over-provisioned space without constraining our problem to traditional RAID geometries. We propose solving this problem quantitatively by mapping it to an innovative variation of the n-queens problem that utilizes a 3D Latin board configuration [12,16], nontraditional queen types and attack domains, and population limits on the number of queens of a given type placed within certain bounds.

The challenge of defining dynamic syndromes is inherently characterized by a well-defined set of constraints: total number of disks, current disk utilization, distribution of unutilized space, existing dependence relationships due to RAID reliability syndromes, and deduplication relationships. By creating a mapping to n-queens under these constraints, we can intuitively represent the problem in a way that facilitates validation and harness the power of the Satisfiability Modulo Theories (SMT) solver Z3 to return a constraint-satisfying solution or determine that a solution cannot exist. Z3 [6] is a very efficient and freely available solver for SMT, which is a decision problem for logical first order formulas with respect to combinations of background theories including the uninterpreted functions integral to our solution. The n-queens problem is a classic way to represent such a constraint satisfaction problem [17,25] and a common benchmark for such a solver [13]. Classification as a constraint satisfaction problem that can be solved by Z3 has proven to be successful in other design domains, such as automating design of encryption and signature schemes [1].

Our contributions include a new quantitative solution for the problem of dynamic allocation of new reliability syndromes while respecting dependence constraints to improve the reliability of software defined data centers without the addition of new hardware. We define and prove a correct mapping of this problem to a variation of the classic n-queens problem, thus enabling efficient analysis via powerful SMT solvers like Z3. We provide an implementation in Z3 for python and include a case study demonstrating the effectiveness of our technique. This new solution will serve as the core for a dynamic allocation system to be deployed in software-defined data centers which will be deployed at the laboratories of partner organizations.

This paper is organized as follows: Sect. 2 provides background on dependence relationships in storage systems, and related work in novel RAID geometries. Section 3 introduces an encoding for this problem in a variant of n-Queens mapping the problem of data layout strategies which respect all data dependence



(a) Example of a virtual disk being constructed out of overprovisioned space.

(b) Example of independent syndrome calculation as the XOR parity of diagonals.

Fig. 1. Example allocations of virtual disks from over-provisioned space.

contraints to maximize additional syndrome coverage for any given dataset to the problem of placing novel queen-types on a Latin chess board; we discuss the constraints of our problem in Sect. 4, and its implementation in Z3. We provide experimental results demonstrating the efficacy and efficiency of our approach in Sect. 5. Finally, Sect. 6 concludes and points to future work.

2 Characterizing File System Dependence

As we have shown in our previous work [3,23], it is possible to predict the future storage resource needs of the users in a system. In recent work [3], we have modeled user behaviors using the training data we obtained from a real system to create and train Markov models, and predicted the future disk usage needs of the users in an on-line fashion, and compared the results with the test data we also obtained from the same system to measure the prediction performance. We have observed that with a good clustering method and fine parameter tuning, it is possible to predict user behaviors and resource requirements. We have proposed this method for predicting over-provisioning, and further that it could allow for dynamic improvement of reliability through the allocation of additional

Table 1. Annual rates of block loss (ABL) per system type with varying numbers of additional syndromes (n_{synd}) allocated.

RAID5 conf.	ABL (no syndromes)	ABL ($n_{synd} = 2$)	ABL ($n_{synd} = 3$)
5+1	$1.79\text{x}10^{5}$	$1.31\text{x}10^{-7}$	$1.92\text{x}10^{-15}$
8+1	$4.60\text{x}10^{5}$	$1.02\text{x}10^{-6}$	$5.06\text{x}10^{-14}$
10+1	$8.06\text{x}10^{5}$	$2.76\text{x}10^{-6}$	$1.79\text{x}10^{-13}$

syndromes by creating new *virtual disks* using any over-provisioned storage that are found to be independent of the current RAID grouping as shown in Fig. 1a. Our experiments on real storage system data have shown that even when being incredibly risk adverse, we can allocate between three and four additional syndromes more than 50 % of the time, and on average allocate two additional syndromes for all of the data and three additional syndromes for more than 90 % of the data, dramatically improving the reliability of the system [3]. We analyzed these improvements on systems with one petabyte of primary storage with initial RAID5 configurations of 5+1, 8+1 and 10+1 over which we introduce two and three additional syndromes after predictions. Changes in reliability are measured using the rate of annual block loss (ABL), when taking into account whole disk failures and latent sector errors. Table 1 illustrates the calculated ABLs for three RAID5 configured primary storage systems, each provisioned for a maximum capacity of one petabyte. The steep increase in the reliability represented by decrease in ABL rates as the number of allocated syndromes increases shows the promise of such predictive analysis and dynamic allocation.

Allocation of new syndromes in order to increase the reliability is possible through the deployment of RAID5 XOR parity syndromes [20] or RAID6 Galois-field based syndromes [2,5]. Additional syndromes can be allocated using techniques such as erasure coding, though they generally have a severe impact on performance, and as a result, lower the QoS of the system [15]. As such, we focus on alternative RAID geometries to make use of additional XOR parity and Galois-field based syndromes.

In this paper we propose an efficient method for allocation of additional syndromes. Additional coverage can be provided using non-traditional RAID geometries as shown in Fig. 1b. While the idea of using non-traditional RAID geometries itself is not new, and have been explored in previous studies [18,19,27], prior work in this field has always maintained the assumption that the layout of the RAID arrays are pre-defined. Instead, we propose the creation of dynamic per-stripe geometries using over-provisioned space in an existing data center.

When creating non-traditional RAID geometries, care must be taken to respect data dependence relationships [24] to ensure that the new RAID strategy improves reliability. We consider two types of data dependence relationships; one resulting from pre-existing RAID groups, and the other from data deduplication [22].

2.1 Dependence Due to RAID

In order to improve reliability, a syndrome allocated as part of a RAID group must be independent of all other syndromes computed within the data center. There are two primary methods for establishing independence with respect to a syndrome, and a set of data on disks. First and most simply, we consider a data center D to be a collection of disks $D = \{d_0, d_1, d_2, d_3, \ldots\}$ protected by a set of RAID strategies $R = \{R_0, R_1, R_2, \ldots\}$ each of which is represented as a set of M disks $R_k = \{d_{k,0}, d_{k,1}, d_{k,2}, \ldots d_{k,(M-1)}\}$ forming a *stripe*. Each stripe consists

of a number of data and syndrome disks. For any two disks $d_i \in D$ and $d_j \in D$ there is a mutual dependence relationship $\leftrightarrow$, $d_i \leftrightarrow d_j$ if $\exists R_k \in R$ such that $d_i \in R_k$ and $d_j \in R_k$. If no such R_k exists then d_i and d_j are independent with respect to RAID.

2.2 Dependence Due to Deduplication

Data deduplication is a wide spread technique which improves the storage efficiency of a data center by eliminating redundant data. As shown in Fig. 2 deduplication identifies substrings of redundant data, typically between block boundaries, stored in a data center and replaces the instances of all but one of those substrings with a reference to a single stored instance on another disk. This creates a critical dependence relationship as the loss of that disk, which may be on another RAID group, may render the data which references it useless, resulting in correlated failures. As such data deduplication introduces one way dependence relationships which we represent as $d_i \rightarrow d_j$ when d_i depends on d_j. Given a disk d_i if $\exists d_l \in R_k$ such that $d_i \in R_k$ and d_l contains a reference to deduplicated data on disk d_j then the dependence relationship $d_i \rightarrow d_j$ exists, otherwise d_i is independent of d_j with respect to deduplication.

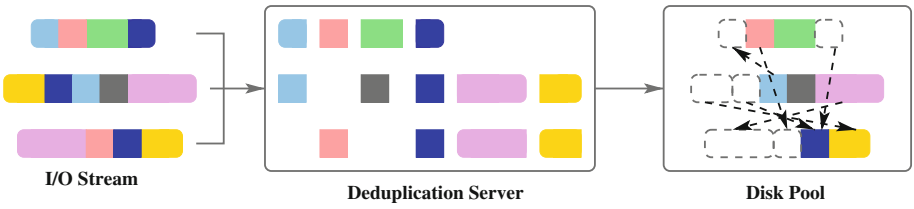


Fig. 2. Data deduplication identifies similar blocks within the data center, and eliminates redundant information.

3 *N*-Queens with Dynamic Domains of Attack

In order to solve our problem and find a data layout that allows us to build virtual disks which are independent of the data they are protecting, we provide a mapping of our problem into a variant on the classical n-Queens [10] constraint satisfaction problem with few alterations.

First, we adopt a Latin board, allowing us to examine our problem in a three dimensional space [12,16]. We define this space according to three axes, the level, rank, and file, as shown in Fig. 3. We further define a column on this Latin board as the set of squares defined by a fixed rank and file across all levels of the board. Each column in our Latin board corresponds to a disk within our data center, with each rank consisting of a traditional RAID group. Levels represent independent sub-problems solving for data independence for each disk in turn. Thus, in practice, given a problem with N ranks and M files, we construct our board with $L = N \cdot M$ levels.

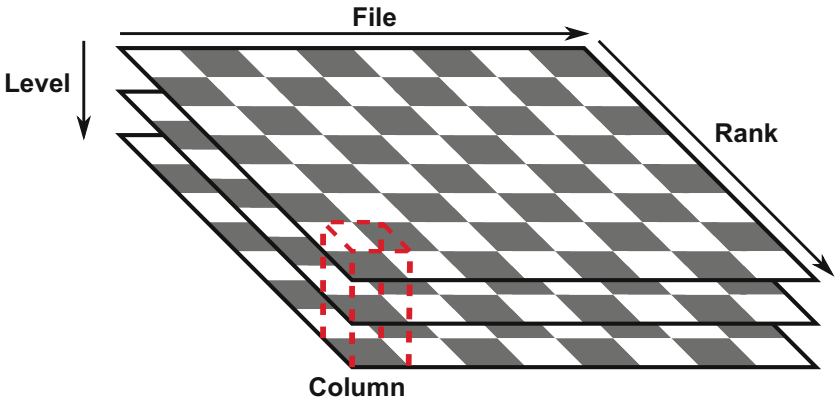
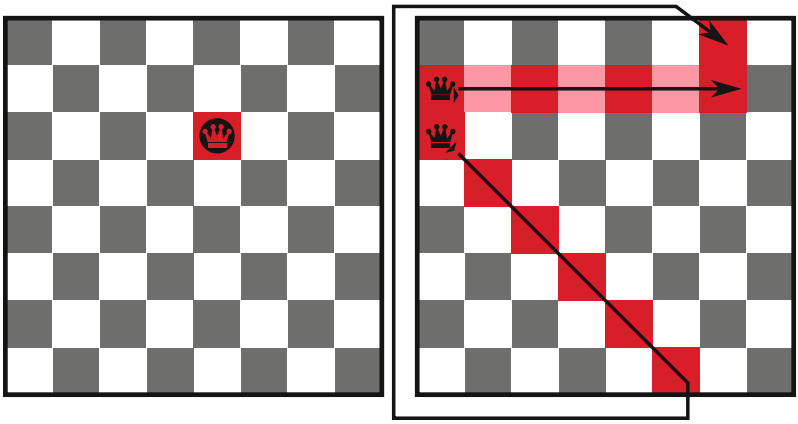


Fig. 3. Example Space with dimensions $3 \times 8 \times 8$.

We represent the state of dependence relationships in a file system by placing Queens on our boards, using their attack domains to represent file dependence relationships. For any level l in our board, this level is used to solve a sub-problem for the lth disk in our data center (numbered in rank major order, such that if the disk is in rank r and file f the level which solves its independence constraints is $l = r*M+f$) the full attack domain of all queens on level l represents those disks on which the lth disk depends. We call this lth disk for level l the principle disk for that level. To represent these dependence relationships, however, we must modify the attack domain definitions for each queen to match the dependence relationships we must represent. We introduce three new queen types each with a unique attack domain.

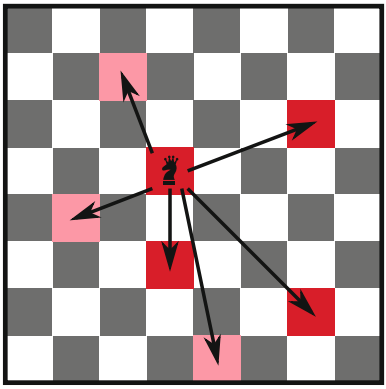
- **Degenerate Queens** - a degenerate queen is so named because it attacks only a single square, that which it is occupying. Degenerate queens are used to represent the disk being protected, and disks containing deduplicated blocks upon which files on that disk depend. Degenerate queens are used to exclude a square on a level from the solution space of new dynamic RAID groupings. The attack domain of a degenerate queen is illustrated in Fig. 4a.
- **Linear Queens** - a linear queen's attack domain is defined to include both its own square and $M-2$ squares on the board extending in a line from the queen, potentially wrapping around the board as if it were a toroidal board as discussed originally in the class of Modular n-Queens problems [9]. Linear queens can be used to represent existing RAID groups, or new dynamic RAID groups with more traditional geometries. Two example attack domains for linear queens are illustrated in Fig. 4b[1].

[1] While we allow linear queens to attack in any direction as a matter of completeness of our variant n-Queens definition, we note that our method only makes use of linear queens which attack along ranks towards squares in higher numbered files, wrapping toroidally.



(a) Example attack domain of a single degenerate queen.

(b) Example attack domain of two linear queens.

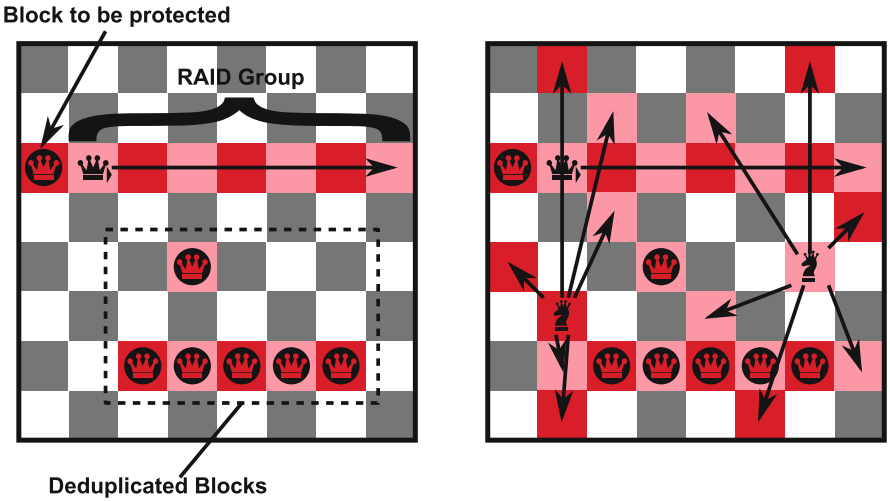


(c) Example attack domain of a single indirect queen.

Fig. 4. Example attack domains of all three queen types.

– **Indirect Queens** - the final type of queen we introduce is an indirect queen, whose attack domain consists of its own square, and $M - 2$ other squares on the board, each within a rank unique to the queen's attack domain. The attack domain of an indirect queen is illustrated in Fig. 4c.

In order to solve the problem of independent syndrome placement, and the creation of new dynamic RAID groupings, we begin with a pre-defined board based on the state of the data center, which contains a number of degenerate and linear queens representing this system state, such as the example shown in Fig. 5a. We then proceed to place Q new indirect queens on the board with each indirect queen representing the storage location of a new pair of XOR and galois field parity syndromes, and the attack domain of that queen representing the independent disks to use to form a new dynamic RAID group associated with those syndromes.



(a) Example of initial constraints when protecting a block on disk 0 of RAID group 2 which has references to deduplicated blocks on six other disks. Degenerate queens are used to include the disks containing the initial and deduplicated blocks in the attack domain, and a linear queen is used to include the RAID group in the attack domain.

(b) An example solution with two additional syndromes. Indirect Queens occupy the spaces corresponding to the disks in which the new syndromes will be stored, their attack domains include all disks protected by the new syndrome.

Fig. 5. Representation of a single level of an 8×8x64 board.

The initial placement of degenerate and linear queens is accomplished according to the dependence relationships defined in Sect. 2. We begin by placing a single degenerate queen on the square corresponding to our level and the disk it solves for, d_i, i.e. for level l we place a degenerate queen at $(\frac{l}{M}, l \mathrm{mod} M)$. If we placed that degenerate queen in the last file of it's rank we place a linear queen at $(\frac{l}{M}, 0)$, otherwise we place it at $(\frac{l}{M}, (l \mathrm{mod} M) + 1)$. This linear queen accounts for the mutual RAID dependence relationships. We then place a single degenerate queen on each square corresponding to some d_j for which $d_i \rightarrow d_j$ to account for the data deduplication derived dependence relationships.

Once the initial board is fixed for each level, we attempt to solve for placing indirect queens representing new RAID groups such that neither the new queens, nor their attack domains overlap with the existing attack domains on the given level. An example solution with two additional syndrome storage locations (allowing up to four additional syndromes) is shown in Fig. 5b.

Our final modification to the traditional n-Queens problem is the addition of a population constraint board. This board enforces a limit to the number of indirect queens placed on any level in a given column, constraining the possible satisfying assignments which respect dependence relationships, and possibly rules some columns out entirely when it comes to queen placement.

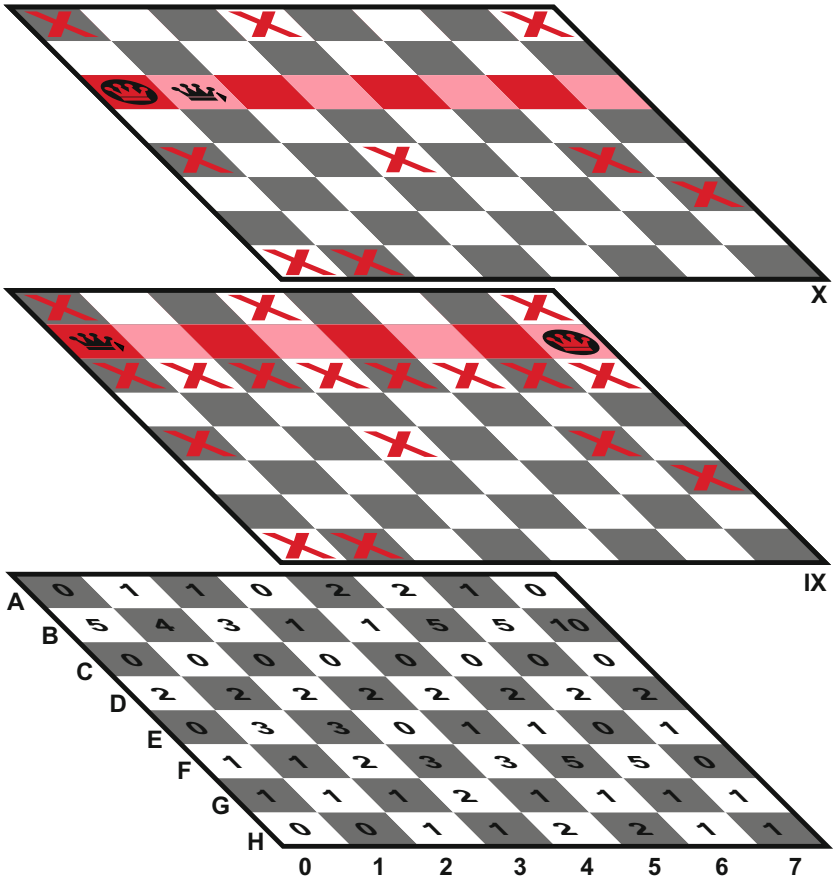


Fig. 6. Example of a Problem with a Population Constraint Board. This board is unsatisfiable for any indirect queen placement due to the population constraints present in rank C. While a placement may exist for level X, no placement can exist for level IX as we need to place an indirect queen and allocate six attack domains, all on different ranks. Note only two levels are shown here, when in a full problem a total of 64 levels would exist.

Figure 6 illustrates an example of a population constraint board being used with a $2x4x4$ Latin board. These population constraints represent the overprovisioned space on each disk that is available for additional reliability syndrome creation as estimated by our predictive models.

While the less restrictive attack domains of our three new queen types would seem to make the problem less difficult than traditional n-Queens, and more equivalent to the trivial n-Rooks problem [4,29], the population constraints board serves to complicate the problem of queen placement, especially as the number of levels we must solve for grows polynomially. The population constraints board has the effect of creating attack domains in the z-axis when enough queens are placed in a column. Figure 7 shows the relative difficulty of

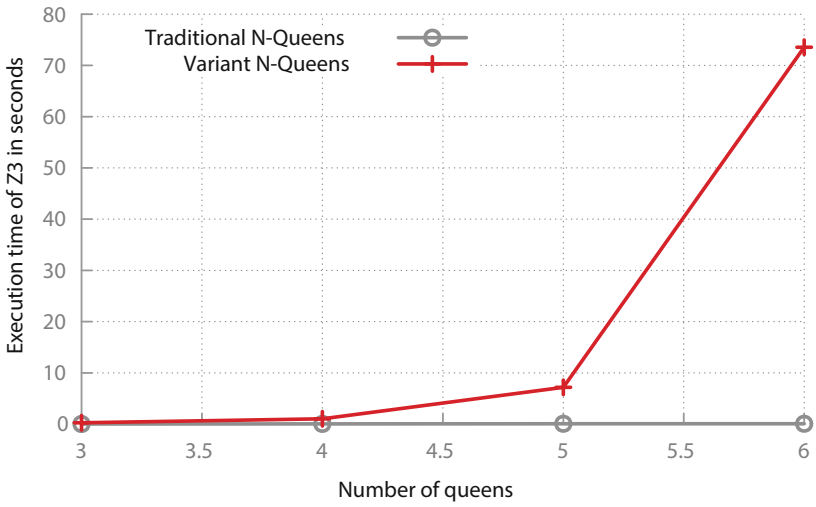


Fig. 7. Comparison of solution times for Z3 given the placement of Y queens on a $Y \times Y$ traditional n-Queens board and a $Y^2 \times Y \times Y$ variant n-Queens board from our problem.

solving this new variant n-Queens problem vs. traditional n-Queens, and highlights the additional complexity despite the more easily satisfied attack domains of our variant queens. While this graph suggests that scalability is an issue, we will address scalability concerns in Sect. 5 through a proposed compositional approach.

4 Solving Virtual Disk Allocation with *N*-Queens

Given the encoding of our problem into n-Queens, to solve for the placement of indirect queens representing new virtual disks holding unique syndromes we define a set of constraints which maintain the necessary independence relationships to ensure our new RAID groups provide additional reliability. In total we can encapsulate these requirements in five constraints.

Constraint 1 (Standard n-Queens). *No queen may be placed within the attack domain of an existing queen.*

As in standard n-Queens problems no indirect queen may be placed within the attack domain of an existing queen. Since we define the attack domain of an existing queen to also encompass the square in which the queen is placed, no two queens may occupy the same square on a board. This ensures that we do not attempt to allocate a new syndrome on a disk which the principle disk for the level is dependent.

Constraint 2 (Non-intersecting Attack Domain). *The attack domains of any two queens may not intersect.*

In addition to the traditional n-Queens requirement that no queen may attack another queen, we further constrain the problem with the rule that for a given level l no queen may be placed such that the attack domain of that queen intersects with the attack domain of an existing queen. Since the attack domains of indirect queens represent independent disks which will be used to produce a new RAID group, and the existing attack domain of all queens on a given level represent all disks which the principle disk for that level depends on, this rule is necessary to produce a new RAID group which will provide additional reliability.

Lemma 1. *Constraints 1 and 2 ensure new syndromes are allocated in independent space on the disk with respect to the block for which it will provide additional reliability.*

Proof. The set of dependencies for a given block already allocated on a system are described in total by the current queens on the appropriate board, and their attack domains. The block to be protected, and any deduplicated blocks will be represented by degenerate queens, thus constrained by 1, the default RAID grouping for the block in question will be represented by the attack domain of a linear queen, thus constrained by 1, and any additional dynamic syndromes will be represented by the attack domain of an indirect queen, also constrained by 1.

Constraint 3 (Indirect Queen Attack Domain). *Each element of a given indirect queen's attack domain must be on a unique rank.*

We assume that our system contains a number of initial RAID groupings corresponding to each rank in our system. As such given two disks d_i and d_j, if both are of the same rank it must be true that $d_i \leftrightarrow d_j$. As such to form a new RAID group of strictly independent disks, a single indirect queen must not contain two square within the same rank in its attack domain.

Lemma 2. *Constraint 3 guarantees that a new indirect queen will be an independent grouping allowing XOR or Galois parity protection.*

Proof. In order for independent XOR or Galois parity blocks to be computed every element of a RAID grouping must be independent. If two attack domains for an indirect queen are on the same rank, they will not be independent, as they will already be involved in another parity group protecting each other.

Constraint 4 (Column Indirect Queen Population Limit). *Each column may be assigned a population limit $p_z \in \mathbb{N}$. The total number of all indirect queens within that column must not exceed this limit.*

As defined in Sect. 3 we include a population constraint board to represent limits in available over-provisioned space in our data center. These population limits restrict the placement of queens within a given column, creating dependence between all levels of our Latin chess board.

Constraint 5 (Level Protection Requirement). *For a given level l, the sum of the number of all indirect queens on that level must be greater than or equal to the protection requirement P.*

As we want to ensure uniform protection of all queens, we constrain the solutions for our system to those which have equal queens on all boards and find the maximum P for which our problem is satisfiable.

4.1 Translation into Z3

In order to determine if a given data center state and desired protection level is satisfiable, we utilized Z3 and encoded our problem in the form of variables and uninterpreted functions forming an SMT problem. We first defined N^2M^2 integer variables, one for each square in our Latin n-queens board with domains of $\{-1, 0, 1, 2\}$ representing:

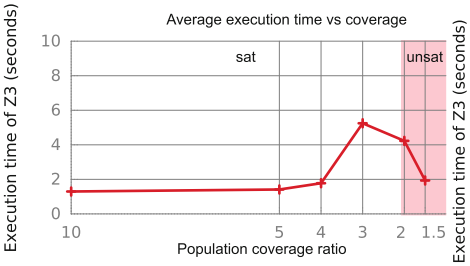
-1 : The attack domain of an indirect queen
0 : A degenerate queen, a linear queen, or the attack domain of a linear queen.
1 : An indirect queen.
2 : An empty square.

For the initial board setup we included an uninterpreted function fixing the assignment of the corresponding variables to 0. Population limits were enforced by the inclusion of uninterpreted functions which constrained the count of variables assigned a value of 1 in a given column to the limit for that column (fed as input to the solver) using the `If()` function of Z3. A protection level of P was enforced for each level by ensuring the count of variables assigned a value of 1 on a level was equal to P using the `If()` function of Z3. We ensured that each indirect queen could be assigned an appropriate attack domain by ensuring the count of variables assigned a value of -1 on a level was equal to $(M-2)P$ using the `If()` function of Z3, and that the count of variables assigned a value of -1 in a given rank for a given level was less than or equal to P. These last two uninterpreted functions also served to make solution more efficient by accounting for symmetric and lump-able attack domains in a single solution [7].

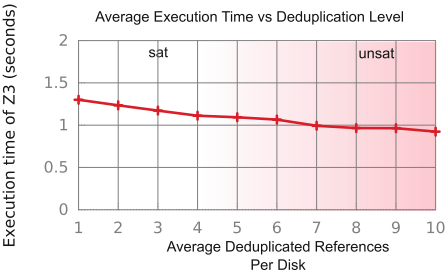
5 Experimental Results and Validation

In order to validate our results we conducted experiments with random initial system states for both population constraints boards, and data deduplication constraints. All experiments were run using a single EC2 c4.large instance with 2 virtual CPUs and 3.75 GiB of RAM. We implemented our solver to print out the resulting boards in a human readable format and hand checked the results, also collecting performance statistics for the Z3 solutions.

Figure 8 along with Fig. 7 provide a summary of the results of our experiments. We found a sharp satisfiability cliff accompanying the population constraint's board which corresponded to the probability of a rank having no available space. This suggests an important observation to account for when moving forward with



(a) Run time and satisfiability of random problems as the population constraints board is made more restrictive.



(b) Run time and satisfiability of random problems as the deduplication ratio is increased.

Fig. 8. Partial summary of experimental results.

a full implementation of software defined data centers, namely that balancing of over-provisioned space can be critical when such space becomes rare and the data center approaches capacity if the excess space is to be used to improve reliability. This limit is approached even swifter for large systems in which many levels are competing for the same population constraints within a rank.

We found the problem to be less sensitive to deduplication. While we eventually found a region of unsatisfiable problems at higher deduplication ratios, the more random placement of deduplicated references ameliorated their constraints on the solution space. It should also be noted that such constraints only became an issue at very high levels of deduplication, suggesting that deduplication based dependences are not as difficult to account for as might be expected.

The exponential growth in runtimes is somewhat concerning, as it seems to limit this solution technique to smaller storage systems, which presents a problem when confronted with the exponential growth of Big Data. Large-scale systems could potentially take infeasible amounts of time to solve if solved directly. As a consequence of this result we propose that larger systems be solved compositionally. For instance, while a 160 TB system takes 74 s to solve, if the system is blocked into two 80 TB systems by decomposing individual ranks a satisfying solution for each system can be found within 2.5 s each, and can be solved in parallel. The exponential improvements found through compositional solution, coupled with the embarrassingly parallel nature of the SMT sub-problems created by partitioning the system by rank provides a very scalable alternative to attacking the entire problem at once. This method has the advantage of respecting dependence relationships, as when decomposed into separate sub-problems all relationships can be accounted for between sub-models in a trivial fashion since their proposed solutions will include only those ranks within a given subproblem.

Since the population constraint board is known as part of the system state, we can choose to sort each rank into one of S subproblems based on the rank of the population constraint board associated with the rank of the Latin board. The satisfiability of the subproblems, depending primarily on these population constraints, can be maximized by sorting the ranks on the basis of the population

constraints associated with their columns. Using such a solution we are able to scale linearly with the size of our data center.

6 Conclusions

In this paper we have presented a novel formulation of the n-Queens problem using a moudlar Latin board, non-traditional queen variants, and column-based population constraints. This formulation serves as a translation of data dependence constraints and the problem of virtual syndrome creation for software-defined data structures into SMT allowing for efficient solution which allows for improved reliability with no additional hardware in over-provisioned systems. While our problem grows exponentially more difficult for larger storage systems, we provide a scalable way to achieve similar levels of protection through rank-wise decomposition of the problem space using population-constraint sorting into embarrassingly parallel subproblems.

This new method will form the basis for a performable dynamic RAID allocation system for use in large-scale storage systems serving cost-constrained organizations, providing an intelligent software stack which will help to combat the exponential growth of Big Data.

6.1 Future Work

Now that we have an efficient, scalable method for determining whether there exists a dynamic reliability syndrome that satisfies its data dependence constraints, we can move onto looking at other interesting optimizations. Currently we either generate a single strategy for additional syndrome allocation, or prove that no such allocation exists. However, the option is now open for us to harness more of the power of Z3 to query the solution space to optimize for secondary considerations, such as geometries that we find more attractive. For example, we may search for solutions with such features using the solution enumeration capabilities of Z3 [14].

We plan to implement our solution technique in a hardware-based middleware controller that monitors back-end data systems, and reshapes incoming file traffic to build the proposed dynamic allocations of RAID groups in response to predictions for overprovisioning. We can also envision an extension enabling data storage system designers to query Z3 regarding hypothetical disk configurations and data dependence constraints as they design a new storage system, thus enabling them to optimize their designs with respect to the robustness/cost tradeoff before purchasing any hardware.

Availability. We have made our implementation, all associated source code, and data available under the terms of the University of Illinois/NCSA Open Source License[2] at our laboratory website http://trust.dataengineering.org/research/nqueens/.

[2] http://opensource.org/licenses/NCSA.

References

1. Akinyele, J.A., Green, M., Hohenberger, S.: Using smt solvers to automate design tasks for encryption and signature schemes. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pp. 399–410. ACM (2013)
2. Anvin, H.P.: The mathematics of raid-6 (2007)
3. Bayram, U., Rozier, E.W., Zhou, P., Divine, D.: Improving reliability with dynamic syndrome allocation in intelligent software defined data centers. In: IEEE/IFIP International Conference on Dependable Systems & Networks, DSN 2015. IEEE (2015)
4. Bell, J., Stevens, B.: A survey of known results and research areas for n-queens. Discrete Math. **309**(1), 1–31 (2009)
5. Corbett, P., English, B., Goel, A., Grcanac, T., Kleiman, S., Leong, J., Sankar, S.: Row-diagonal parity for double disk failure correction. In: Proceedings of the 3rd USENIX Conference on File and Storage Technologies, pp. 1–14 (2004)
6. de Moura, L., Bjørner, N.S.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
7. Derisavi, S., Kemper, P., Sanders, W.H.: Symbolic state-space exploration and numerical analysis of state-sharing composed models. Linear Algebra Appl. **386**, 137–166 (2004)
8. Duffy, D., Schnase, J.: Meeting the big data challenges of climate science through cloud-enabled climate analytics-as-a-service. In: Proceedings of the 30th International Conference on Massive Storage Systems and Technology. IEEE Computer Society (2014)
9. Eickenscheidt, B.: Das n-damen-problem auf dem zylinderbrett. Feenschach **50**, 382–385 (1980)
10. Gu, J., et al.: Efficient local search with conflict minimization: a case study of the n-queens problem. IEEE Trans. Knowl. Data Eng. **6**(5), 661–668 (1994)
11. Hashem, I.A.T., Yaqoob, I., Anuar, N.B., Mokhtar, S., Gani, A.B., Khan, S.U.: The rise of big data on cloud computing: review and open research issues. Inf. Syst. **47**, 98–115 (2015)
12. Klarner, D.A.: Queen squares. J. Recreational Math **12**(3), 177–178 (1979)
13. Klebanov, V., et al.: The 1st verified software competition: experience report. In: Butler, M., Schulte, W. (eds.) FM 2011. LNCS, vol. 6664, pp. 154–168. Springer, Heidelberg (2011)
14. Köksal, A.S., Kuncak, V., Suter, P.: Scala to the power of Z3: Integrating SMT and programming. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS, vol. 6803, pp. 400–406. Springer, Heidelberg (2011)
15. Leventhal, A.: Triple-parity raid and beyond. Queue **7**(11), 30 (2009)
16. McCarty, C.P.: Queen squares. Am. Math. Monthly **85**, 578–580 (1978)
17. Nadel, B.A.: Representation selection for constraint satisfaction: a case study using n-queens. IEEE Intell. Syst. **5**(3), 16–23 (1990)
18. Pâris, J.F., Amer, A., Schwarz, T.J.: Low-redundancy two-dimensional raid arrays. In: 2012 International Conference on Computing, Networking and Communications (ICNC), pp. 507–511. IEEE (2012)
19. Pâris, J.F., Long, D.D., Litwin, W.: Three-dimensional redundancy codes for archival storage. In: 2013 IEEE 21st International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 328–332. IEEE (2013)

20. Patterson, D.A., Gibson, G., Katz, R.H.: A case for redundant arrays of inexpensive disks (RAID), vol. 17. ACM (1988)
21. Pless, V.: Introduction to the Theory of Error-Correcting Codes, vol. 48. Wiley, New York (2011)
22. Rozier, E.W., Sanders, W.H., Zhou, P., Mandagere, N., Uttamchandani, S.M., Yakushev, M.L.: Modeling the fault tolerance consequences of deduplication. In: 2011 30th IEEE Symposium on Reliable Distributed Systems (SRDS), pp. 75–84. IEEE (2011)
23. Rozier, E.W., Zhou, P., Divine, D.: Building intelligence for software defined data centers: modeling usage patterns. In: Proceedings of the 6th International Systems and Storage Conference, p. 20. ACM (2013)
24. Rozier, E.W.D., Sanders, W.H.: A framework for efficient evaluation of the fault tolerance of deduplicated storage systems. In: 2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 1–12. IEEE (2012)
25. Salido, M.A., Barber, F.: How to classify hard and soft constraints in non-binary constraint satisfaction problems. In: Coenen, F., Preece, A., Macintosh, A. (eds.) AResearch and Development in Intelligent Systems XX, pp. 213–226. Springer, New York (2004)
26. Schnase, J.L., Duffy, D.Q., Tamkin, G.S., Nadeau, D., Thompson, J.H., Grieg, C.M., McInerney, M.A., Webster, W.P.: Merra analytic services: Meeting the big data challenges of climate science through cloud-enabled climate analytics-as-a-service. Computers, Environment and Urban Systems (2014)
27. Schwarz, S., Long, D.D., Paris, J.F.: Reliability of disk arrays with double parity. In: 2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 108–117. IEEE (2013)
28. Turner, V., Gantz, J.F., Reinsel, D., Minton, S.: The digital universe of opportunities: Rich data and the increasing value of the internet of things. International Data Corporation, White Paper, IDC_1672 (2014)
29. Weisstein, E.W.: Rooks problem (2002)