SMT-Driven Intelligent Storage for Big Data

Eric W. D. Rozier and Kristin Y. Rozier
University of Cincinnati - Ohio, USA
Email: {Eric.Rozier, Kristin.Y.Rozier}@uc.edu

Abstract—**Big Data presents itself as a double-edged sword: both promising and problematic. While large-scale data collection and analytics are revolutionizing scientific discovery, they also pose serious challenges for infrastructure in cost-constrained environments. NASA's Center for Climate Simulation named data infrastructure the biggest problem facing climate science, with the growth of storage needs outstripping the growth of necessary computing power by 6.67x. New advances in intelligent software-defined data centers (SDDCs) have the potential to combat these challenges, but they must incorporate the ability to optimize data dependence constraints quickly and performably. We formally define data dependence constraints for Big Data storage architectures, present automated methods for encoding the system states of such architectures into constraints suitable for SMT solving with Z3, and describe methods for efficiently resolving our data dependence constraints to enable SDDCs. We include proofs of correctness of our new constructions and optimizations inspired by experimental evaluation.**

I. Introduction

The exponential growth of Big Data [1] presents both an opportunity for disruptive innovations in many scientific fields, and a tremendous challenge for system architects. For example, today's resource-constrained environments provide a particular challenge for advancements in aerospace as, "the bisectional bandwidth of future avionics platforms could easily exceed Petabytes with video-based applications and possibly reach the capacity of small data centers" [2]. Unmanned Aerial Systems (UAS) are one of today's hottest new-technology domains yet the storage redundancy required for their safety-critical applications exceeds size, weight, price, and power consumption limits [3], making the system too complex and costly to fly. Space avionics vitally require fault-tolerant data storage due to radiation and harsh operating environments; combined with their very long mission lifecycles, "strategic storage" is essential to reliably capture the big data produced [4]. Indeed, NASA's Solar Observatory produces over 1,600 gigabytes of data each day [5]. Collecting large volumes and varieties of data is revolutionizing many areas of scientific inquiry, including healthcare [6], genomics [7], urban-planning [8], climate-science [9], economics [10], and more.

The counterpoint to the increase in our analytical and predictive ability is the difficulty in dealing with the scale of Big Data. According to the NASA Center for Climate Simulation (NCCS), while computing needs have risen by a factor of 300 in the last ten years, storage needs have risen by a factor of 2,000 [9]. The NCCS called storage infrastructure one of the largest challenges facing climate scientists, and the case is similar across engineering. Moreover, Big Data centers are increasingly cost-constrained, e.g., by declining federal funding, and must do more with fewer resources.

The authors thank Ulya Bayram for prior work implementing Z3 constraints in Python; and Varun Krishna and Rohit Dureja for helpful comments.

All artifacts required for full reproducibility are available at http://dataengineering.org/research/CFV2015.

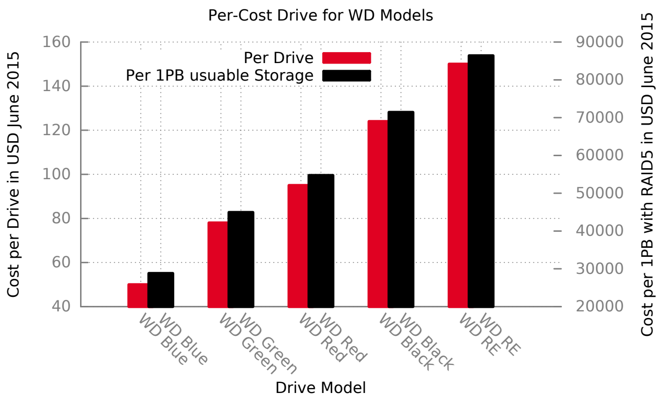

Fig. 1: Comparison of the cost per drive and per PB of storage for several models of Western Digital hard drives.

Fig. 1 shows hard drive costs as of Summer 2015[1]. While reliance on enterprise-class drives was once common due to the additional reliability supposedly afforded, purchasing 1PB of enterprise-class drives can cost more than $86,000. By comparison, the same budget will purchase more than 3PB of storage using near-line drives. Other costs are inherent to enterprise-class systems as well, including purchasing specialized motherboards, controllers, and power supplies, and hiring dedicated support staff with the appropriate skill sets.

Another characteristic of modern storage systems in scientific and non-profit environments is the departure from industrial best practices on thin provisioning: only enough storage is purchased, installed, and configured to account for the most immediate needs. This departure is motivated by two primary factors. First is the **impractical** nature of thin-provisioning. Big Data is often characterized by "bursty" behavior in which data arrives in large bursts followed by long periods of relative inactivity, such as when a spacecraft passes an object of interest, "wakes up" to take a huge volume of observations, and then lays dormant again for years en route to its next observation. These bursty periods have traditionally been difficult to predict. Failure to have enough space provisioned and available can lead to the failure of data delivery and result in either permanent data loss, or possibly high costs associated with recollecting the data if it cannot be staged for sufficient periods of time. Second

[1]Prices collected from Newegg.com and Amazon.com

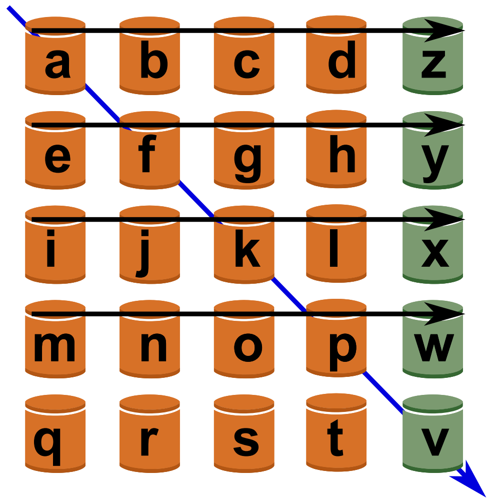

Fig. 2: Example of new independent syndrome creation.

is the **infeasible** nature of thin-provisioning. In many cost- and resource-constrained environments, purchasing decisions are motivated by a feast-or-famine mind-set. Equipment is purchased infrequently in large quantities when the funds are available, followed by long periods in which budgets are limited, or not allocated for equipment. Elastic storage is also infeasible for cost-constrained organizations due to the high cost of ownership of more than \$350,000 per PB per year, more than 10 times the cost of the raw hard drives[2].

These factors combine to produce an environment in which storage systems are over-provisioned for large fractions of their life-cycle, and this over-provisioned space is poorly utilized (if at all). Previous work [11], [12] showed that the actions of users are imminently predictable and models can be generated that can accurately make conservative predictions of the likely future storage needs of a system, even accounting for bursty behavior. Our previous work [13] created a new method for efficiently computing *independent reliability syndromes* that utilize over-provisioned disk space to create dependence relationships in the data that can be exploited for reliability. We demonstrated how to practically reason about large-scale systems via encoding dependencies as a specialized variation of the well-known n-queens problem and using an SMT solver to produce strategies for maximizing reliability. Importantly, this up-to-exponential increase in data reliability comes **at no additional cost**.

In this paper, after some preliminaries in Section II, we extend the results of [13], providing new, formal constraints for that construction in Section III and some helpful proofs related to their correctness, with respect to ensuring independence of new reliability syndromes. We further detail an optimization for our construction in Section III-A, and provide experimental validation in Section IV. Section V concludes with a discussion of future directions.

II. Preliminaries

A typical method for reliability syndrome generation is XOR parity. In a situation such as that shown in Fig. 2, data may be made more reliable by creating a new dependence between currently independent data. So given blocks a, b, c, and d stored on separate physical hardware, a new block $z = a \oplus b \oplus c \oplus d$ can ensure that if any block is lost, for example c, it can be easily recreated as $c = a \oplus b \oplus d \oplus z$, adding to the reliability of the underlying file system [14]. Reliability can be further extended through the use of Galois fields [15], and in theory with erasure codes [16], however codes patent encumbrance has effectively removed performable erasure code algorithms from use [17]. In practice this means for any set of initially dependent data, reliability can be increased (given sufficient space) via creation of two independent syndromes.

Additional reliability syndromes can be allocated using additional blocks not already linked through a syndrome-related dependence, such as a, f, k, and p in Fig. 2. The difficulty inherent in allocating these new syndromes is ensuring independent sets of blocks can be identified, along with independent free space in the storage system. As the storage system becomes fuller over time, the difficulty of this problem increases exponentially, necessitating efficient solution techniques.

We consider three types of data dependence relationships in our analysis. The first are *file dependence* relationships. We consider data in our file system as divided into blocks (typically of fixed size) with each file ϕ being composed of a set of blocks $B_\phi = \{b_{\phi_0}, b_{\phi_1}, \ldots\}$. Blocks that are part of the same file have a file dependence relation represented by $\mathcal{R}(\phi)$ such that for some b_i, b_j, $b_i \mathcal{R}(\phi) b_j$ if there exists some ϕ composed of B_ϕ where $b_i \in B_\phi$ and $b_j \in B_\phi$. Secondly, we consider *reliability dependence*. Such a dependence, represented by $\mathcal{R}(\sigma)$, exists between blocks b_i, b_j if both b_i and b_j participate in reliability syndrome σ. Thus if there exists some σ such that $\sigma = b_i \oplus b_j \oplus \ldots$ then $b_i \mathcal{R}(\sigma) b_j$. Lastly, we consider *deduplication dependence* relationships. These relationships are much like those found in file dependence relationships, and can be defined in the same way, differing only in that for a deduplicated block b_i there can be multiple files in which it participates in dependence relationships, so $b_i \in B_k$ does not preclude that some B_l also exists such that $b_i \in B_l$, $l \neq k$. For convenience, we will also use the notation $\mathcal{R}$ without subscript to indicate the presence of any dependence relationship, regardless of the type. These relationships become important when defining a new syndrome σ' to protect some block b_p. When defining σ' as a set of blocks $\{b_p, b_0, b_1 \ldots\}$ such that $\sigma' = b_p \oplus b_0 \oplus b_1 \oplus \ldots$ it is important to pick blocks such that for $b_i \in \{b_0, b_1 \ldots\}, b_i \mathcal{R} b_p$ is false; otherwise the new syndrome will not provide the expected improvements to reliability as independence is a fundamental assumption for syndrome construction.

We propose a solution to this dependence problem, with the aim of increasing reliability in over-provisioned storage systems without additional hardware. Our solution is based on a new method for encoding system reliability goals as a variation of the n-queens problem; formulating system constraints in this way enables efficient solution via a Satisfiability Modulo Theories (SMT) solver. SMT extends classic

[2]Costs researched via Amazon Web Services; data collected Summer 2015

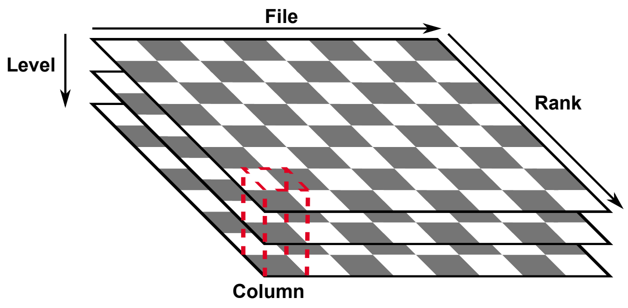

Fig. 3: Dimensions of a Latin squares board.

satisfiability; it is a decision problem for logical first order formulas with respect to combinations of background theories, including the uninterpreted functions integral to our solution. Z3 [18] is an efficient, freely available SMT solver that has been successfully used in industry [19]. We chose it for our implementation because it supports all of the SMT logics while consistently placing highly in nearly all categories of the annual SMT-COMP rankings [20]. Encoding our reliability goals as constraints for SMT solving brings the advantage of efficient, automatic solution constructions. A *satisfiable* assignment from Z3 then represents a strategy for defining SDDCs on our disks that meets our reliability goals; a result of *unsatisfiable* is proof that no such strategy exists.

III. A Formal Problem Representation

In this section we provide a formal representation of our problem accompanied by some helpful proofs, define our representation, and provide constraints for use in SMT solving that allow the production of strategies for reliability improvement, if any such strategy exists. Our solution is intuitive and easy to validate as the n-queens problem is a classic way to represent such a constraint satisfaction problem [21], [22]; n-queens variations are common benchmarks for SMT solvers [23], so it is easy to choose a good solver. We represent our problem in the domain of a Latin-squares [24], [25] variant of n-queens problem (using multiple levels of the Latin-squares board to represent separate, yet dependent, subproblems) using novel variant queen types with unique attack domains, and population constraints. In our representation, the board represents the physical disk media with each column in the Latin-squares board representing a separate physical disk.

Definition 1 (Board)**.** For ease of representation, we utilize a three-dimensional matrix: a Latin squares variant of n-queens with the dimensions L levels, R ranks, and F files, as shown in Fig. 3. A board is a set of $L \cdot R \cdot F$ squares indexed $x_{l,r,f} \in X$. Each square has a variable assignment from a finite set $V = Q \cup A \cup \epsilon$ that represents its state, where Q is a finite set of symbols representing queens of three types, A is a finite set of symbols representing squares under attack by each type of queen, and ϵ is an empty square.

We use the term *column* to indicate a set of variables $X_C \subset X$ where $r = a$, and $f = b$, for some $a \in R$ and some $b \in F$.

Given an array of $R \cdot F$ disks arranged into traditional RAID groupings of F disks per group, each level of the board represents a separate constraint satisfaction problem for a separate set of data associated with a given physical disk. Specifically, $L = R \cdot F$ with level l_i representing the problem associated with disk $r \cdot R + f$, or (r, f). For each set of data on a given physical disk we construct the initial data dependencies by assigning queens and their attack domains to the variables representing a given level.

Queens, and the squares they attack, are used to represent dependence relationships between data on the physical disks represented by the board. We characterize the squares that a queen is said to attack as the *attack domain* of the queen, and the combination of squares that a queen occupies and attacks as the *attack set*.

Definition 2 (Queen)**.** A queen is a symbol from the set $Q = \{\Delta, \Lambda, \Gamma\}$, which can be assigned to any free variable. The three queen symbols differ in their allowed attack domains and are called degenerate queens (Δ), linear queens (Λ), and indirect queens (Γ).

Initial conditions for our data dependence constraints are constructed using two special types of queens, *degenerate queens* and *linear queens* that differ from the standard queens of the classical problem in terms of their attack domains.

Definition 3 (Attack Domain)**.** The attack domain of a queen at position $x_{l,r,f}$ is defined by its position, and a set of additional squares called it's *Attack Set* given by the set $S_{l,r,f}$. This attack domain, $D_{l,r,f} = S_{l,r,f} \cup x_{l,r,f}$, represents every square a queen attacks or occupies.

Definition 4 (Attack Set)**.** An attack set is a set of variables $S_{l,r,f}$ assigned labels from the set $\{\lambda, \gamma\}$ to designate they are attacked by a queen such that $\forall s_i \in S_{l,r,f}, s_i = \lambda$ iff $x_{l,r,f} = \Lambda$, and $s_i = \gamma$ iff $x_{l,r,f} = \Gamma$. Note that there is no attack set associated with a degenerate queen ($x_{l,r,f} = \Delta$) because the attack domain of a degenerate queen contains only the square containing the degenerate queen itself.

Constraint 1 (Linear Queen Attack Set)**.** The attack set of a linear queen assigned to variable $x_{l,r,f}$ must be such that either Constraint 1.1, 1.2, or 1.3 are satisfied. The size[3] of a linear queen's attack set is always equal to $F - 2$ and must satisfy Constraint 1.

Constraint 1.1 (Constant File)**.** The attack set of a linear queen at $x_{l,r,f}$ satisfies the *Constant File* constraint iff $\forall s_i \in S$ the file of s_i is equal to f.

Constraint 1.2 (Constant Rank)**.** The attack set of a linear queen at $x_{l,r,f}$ satisfies the *Constant Rank* constraint iff $\forall s_i \in S$ the rank of s_i is equal to r.

Constraint 1.3 (Unique Rank and File)**.** The attack set of a linear queen at $x_{l,r,f}$ satisfies the *Unique Rank and File*

[3] Both for simplicity and performance reasons, we assume that any additional protection provided uses the same RAID configuration as default RAID groupings. This assumption can be relaxed without loss of generality.

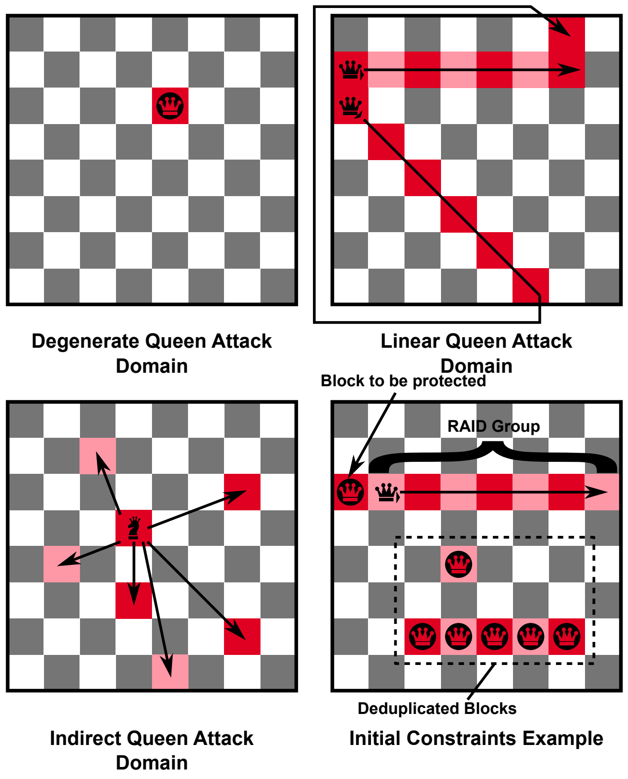
Degenerate Queen Attack Domain
Linear Queen Attack Domain
Block to be protected
RAID Group
Indirect Queen Attack Domain
Deduplicated Blocks
Initial Constraints Example

Fig. 4: Attack domains of the three queen types, and an example of initial constraints of linear and degenerate queens.

constraint iff $\forall s_i, s_j \in S$ the file of $s_i(f_{s_i})$ and $s_j(f_{s_j})$ are such that $f_{s_i} \neq f, f_{s_j} \neq f$, and $f_{s_i} \neq f_{s_j}$, and the rank of $s_i(r_{s_i})$ and $s_j(r_{s_j})$ are such that $r_{s_i} \neq r, r_{s_j} \neq r$, and $r_{s_i} \neq r_{s_j}$.

We then try and find a solution that satisfies all of our constraints, which allows us to place P or more *indirect queens* on our board, where each indirect queen represents a new syndrome to be allocated for reliability, and its attack domain represents the new data dependencies associated with that syndrome.

Definition 5 (Indirect Queen)**.** An indirect queen is represented by Γ and has the corresponding attack symbol γ. The size[4] of an indirect queen's attack set is always equal to $F-2$ and must satisfy Constraint 2.

Constraint 2 (Indirect Queen Attack Set)**.** The attack set of a linear queen assigned to variable $x_{l,r,f}$ must be such that $\forall s_i, s_j \in S$ the rank of $s_i(r_{s_i})$ and $s_j(r_{s_j})$ are such that $r_{s_i} \neq r$, $r_{s_j} \neq r$, and $r_{s_i} \neq r_{s_j}$.

Each indirect queen is able to provide both XOR parity, and Galois field parity for its attack domain, provided the disk has available space. This space constraint holds for an entire column as well, as each column represents a single physical disk. This necessitates the representation of column-wise population constraints in the form of a *population constraint board.*

[4]Making the same assumption as with linear queen attack sets.

Definition 6 (Population Constraint Board)**.** In addition to the defined set of $L \cdot R \cdot F$ variables that make up the board, we add a set P of $R \cdot F$ such that $\forall p_{r,f} \in P, p_{r,f} \in \mathbb{N}$, and call this set of constants the population constraint board.

The population constraint board tracks the available free-space per physical device, and constrains the total placement of indirect queens within a column.

Constraint 3 (Column Indirect Queen Population Limit)**.** Each column may be assigned a population limit $p_{r,f} \in \mathbb{N}$. The total number of all indirect queens within that column must not exceed this limit. We generate $R \cdot F$ new constraints for each combination of unique r and f such that $r \in [0, R-1]$ and $f \in [0, F-1]$

$$\left(\sum_{l \in [0,(R \cdot F)-1]} (x_{l,r,f} = \Gamma) \right) \leq p_{r,f}$$

Constraint 4 (Non-intersection of Attack Domains)**.** In addition to the constraint that no queen fall within the attack domain of another queen, we further constrain the problem by specifying that no attack domain may intersect the attack domain of another queen.

Theorem 1. *Due to our construction and problem representation, Constraint 4 holds for any solution implicitly.*

Proof. From Definition 1 a board is defined by a set of variables, X where variable has a single assignment from V. In a satisfying assignment, no queens attack each other or have overlapping attack domains. Because all variables must have exactly one assignment from V two queens of different types may not both have a square assigned to their attack domains. Thus the only case where two queens might overlap is when they are of the same type. From Definition 5 and Constraint 1 we know that a valid assignment for a board must contain $F-2$ [13] squares in the attack set of any queen, so if there are N queens of a given type on a board there must be $N(F-2)$ squares assigned to their attack domains. If two queens of the same type had overlapping attack domains, fewer than $N(F-2)$ squares would be assigned to the attack domains of those queens, violating Definition 5 and Constraint 1. ☐

We add a level protection requirement as a constraint to specify that all disks are protected by at least W additional syndromes per block on the disk that contains data.

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 W. We generate L new constraints for each $l \in [0, (F * R) - 1]$ of the form

$$\sum_{r \in [0,R-1], f \in [0,F-1]} (x_{l,r,f} = \Gamma) \geq W$$

Theorem 2. *The set of these constraints are both necessary and sufficient to account for all dependencies and restrictions necessary such that a solution to this constraint satisfaction* *problem will be a valid layout for additional independent reliability syndromes on a given file system, and that if no such satisfying solution is found, none exists.*

Proof. In order for a satisfying solution to improve the reliability of the underlying data storage system it must provide at least W additional syndromes per block (i.e. satisfies Constraint 5), must fit within the available over-provisioned space (i.e. satisfies Constraint 3), and for which all new syndromes are independent with respect to the initial data layout and each other (i.e. satisfies Constraint 4). Further more each new syndrome must be built only from independent blocks with respect to the files that contain them (i.e. satisfies Constraint 2) and blocks that are used in other newly allocated syndromes, or which feature dependence on the block for which a given level is solving (i.e. satisfies Constraint 4). □

A. Improving Tractability Through Variable Domain Reduction

Theorem 3. *Our problem representation can be simplified without loss of generality or correctness through the collapse of the variable domain into a smaller subset* $\{\Delta, \Gamma, \gamma, \epsilon\}$.

Proof. The problem can be simplified by realizing that as a consequence of Theorem 1 we can collapse the variable domain V into a smaller set without loss of correctness in our solution. Since we require the SMT solver to return only a placement of indirect queens, and their attack sets, in a way that respects the attack domains of previously placed queens, we need not differentiate those queens from their attack sets, or from each other. □

Definition 7 (Reduction relations on variable assignments)**.** We define a reduction relation over variable assignments.

$$x_{l,r,f} \rightarrow \Delta \quad \text{if} \quad x_{l,r,f} \in A = \{\lambda, \Delta, \Lambda\}$$

Definition 7 allows us to reduce the total set of possible values a variable can be assigned by recognizing that the importance of linear and degenerate queens, and their attack domains, can be reduced to a single value representing a square that a new indirect queen, or it's attack domain, cannot occupy due to Constraint 4

IV. Experimental Results

In order to validate our results we conducted experiments with random initial system states for both population constraints boards, and data deduplication constraints, and with data derived from our partners at the Illinois Natural History Survey and Prairie Research Institute at the University of Illinois at Urbana-Champaign. 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 on select experiments for validation, also collecting performance statistics for the Z3 solutions.

In our experiments the problem showed sensitivity to boards with high probability of a given rank having low square availability, with satisfiability dropping off rapidly, which

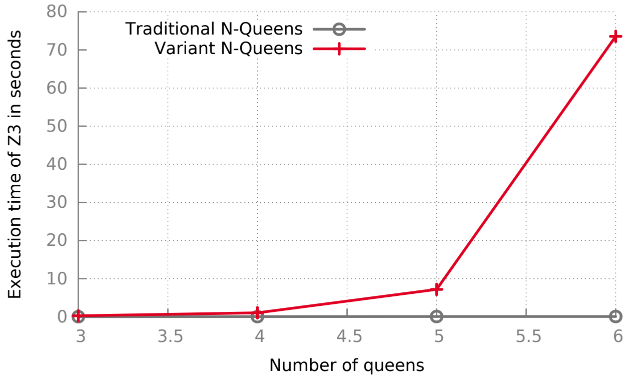

Fig. 5: Comparison of runtimes between traditional n-queens and our variant, vs. the number of queens placed on the board.

suggests the importance of algorithms for balancing over-provisioned space in SDDCs. This limit becomes critical when many levels of a board are competing for the same population constraints within a given rank. The constraint satisfaction problem showed little sensitivity to deduplication, however, with regions of unsatisfiability being smaller, and only occurring with very high deduplication ratios that are uncharacteristic of most real systems. The largely random placement of deduplicated references in practice reduces the probability of an unsatisfiable board. Constraint satisfaction only became an issue at very high levels of deduplication, suggesting that deduplication based dependencies are not as difficult to account for as might be expected.

A. Compositional Solutions

A large potential issue in solving this variant of n-queens is the exponential growth of run times with increasing size of the disk array. As seen in Fig. 5, despite the difference in attack domains for the queens used in our variant, the problem still scales exponentially, and is in fact harder than traditional n-queens. This is primarily due to the population constraints per column, which tie all the boards together creating difficult satisfiability problems for large numbers of queens.

The exponential growth in runtimes is concerning, as it limits this solution technique to smaller storage systems, which presents a problem given the exponential growth of Big Data. Large-scale systems could potentially take infeasible amounts of time to solve if solved directly. We tackle this problem via a transformation of our problem into a series of independent sub-problems that can be solved compositionally. We note that in our experiments with the data provided by INHS that a system with 160TB takes 74 seconds to solve, while a system with two independent 80TB sub-systems can be solved in just 2.5 seconds per subsystem. Compositional solution provides exponential improvements provided a proper decomposition can be found. Such decompositions can be achieved through rankwise slicing of the Latin-squares as shown in Fig. 6, which respects all dependence relationships for traditional RAID geometries. 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 sub-problem.

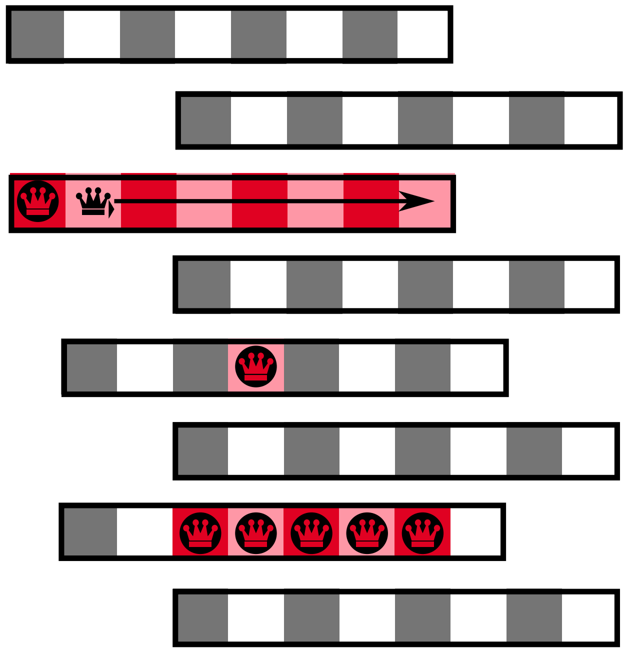

Fig. 6: A rank-separated level, suitable for reshuffling.

Ranks can then be sorted individually based on the column-wise population constraints associated with the rank, allowing new Latin-squares boards to be created with as balanced population constraints as possible to increase the probability of satisfiability of the subproblems. Using such a solution we are able to scale linearly with the size of our data center.

V. Conclusions and Future Work

In this paper we have expanded on our previous work in [13] by formalizing the mathematics behind our unique problem definition, and the constraints inherent in data dependence relationships, forming the crucial groundwork for SMT-based data layout optimization and dependability augmentation in SDDCs. Despite the difficulty inherent in the underlying problem, we provide a variable domain reduction semantic, and compositional solution technique that reduce the difficulty of the problem in practice to maintain feasibility of our approach.

We are working to implement our solution techniques using a hardware-based middleware controller based on Z3. Given Z3's open-source status, it is a more feasible platform for recompilation on specialized architectures and embedded solutions for real SDDCs. This controller will form the basis for a system with the capabilities required to reshape incoming data traffic in order to build the proposed dynamic allocations of RAID groups in response to predictions for over-provisioning. We envision additional extensions enabling data storage system designers to query our controller regarding hypothetical disk configurations and data dependence constraints as they design a new storage system, allowing for the optimization of designs with respect to both robustness of the underlying systems and the resulting cost trade-offs.

References

[1] V. Turner, J. F. Gantz, D. Reinsel, and S. Minton, "The digital universe of opportunities: Rich data and the increasing value of the internet of things," *International Data Corporation, White Paper, IDC_1672*, 2014.

[2] D. J. Blumenthal, "Terabit optical ethernet for avionics," in *Avionics, Fiber-Optics and Photonics Technology Conference (AVFOP), 2011 IEEE*, pp. 61–62, IEEE, 2011.

[3] B. H. Sababha, O. A. Rawashdeh, and W. A. Sa'deh, "A real-time gracefully degrading avionics system for unmanned aerial vehicles," in *Aerospace and Electronics Conference (NAECON), 2012 IEEE National*, pp. 171–177, IEEE, 2012.

[4] M. Pignol, "COTS-based applications in space avionics," in *DATE*, pp. 1213–1219, European Design and Automation Association, 2010.

[5] N. Aeronautics and S. Administration, "Sdo: Solar dynamics observatory." Online: http://www.nasa.gov/pdf/417176main_SDO_Guide_CMR.pdf, 2009. NP-2009-10-101-GSFC; A Guide to the Mission and Purpose of NASA's Solar Dynamics Observatory.

[6] K. Miller, "Big data analytics in biomedical research," *Biomedical Computation Review*, vol. Winter, pp. 14–21, 2012.

[7] D. Howe, M. Costanzo, P. Fey, T. Gojobori, L. Hannick, W. Hide, D. P. Hill, R. Kania, M. Schaeffer, S. St Pierre, *et al.*, "Big data: The future of biocuration," *Nature*, vol. 455, no. 7209, pp. 47–50, 2008.

[8] M. Batty, "Big data, smart cities and city planning," *Dialogues in Human Geography*, vol. 3, no. 3, pp. 274–279, 2013.

[9] J. L. Schnase, D. Q. Duffy, G. S. Tamkin, D. Nadeau, J. H. Thompson, C. M. Grieg, M. A. McInerney, and W. P. Webster, "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.

[10] H. Chen, R. H. Chiang, and V. C. Storey, "Business intelligence and analytics: From big data to big impact.," *MIS quarterly*, vol. 36, no. 4, pp. 1165–1188, 2012.

[11] E. W. Rozier, P. Zhou, and D. Divine, "Building intelligence for software defined data centers: modeling usage patterns," in *SYSTOR*, p. 20, ACM, 2013.

[12] U. Bayram, E. W. Rozier, P. Zhou, and D. Divine, "Improving reliability with dynamic syndrome allocation in intelligent software defined data centers," in *Dependable Systems & Networks, 2015. DSN'15. IEEE/IFIP International Conference on*, IEEE, 2015.

[13] U. Bayram, K. Y. Rozier, and E. W. D. Rozier, "Characterizing data dependence constraints for dynamic reliability using n-queens attack domains," in *Quantitative Evaluation of SysTems (QEST)*, vol. 9259 of *LNCS*, pp. 211–227, Springer, September 2015.

[14] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (raid)," in *Proc. ACM SIGMOD Conf.*, (Chicago, IL), pp. 109–116, ACM Press, June 1988.

[15] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "Raid: High-performance, reliable secondary storage," *ACM Computing Surveys (CSUR)*, vol. 26, no. 2, pp. 145–185, 1994.

[16] R. G. Dimakis, P. B. Godfrey, Y. Wu, M. O. Wainwright, and K. Ramch, "Network coding for distributed storage systems," in *In Proc. of IEEE INFOCOM*, 2007.

[17] C. Henderson, "Usenix association fast 2013 memo." Online: https://www.usenix.org/system/files/conference/fast13/fast13_memo_021715.pdf, 2015.

[18] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *TACAS*, pp. 337–340, Springer, 2008.

[19] J. A. Akinyele, M. Green, and S. Hohenberger, "Using smt solvers to automate design tasks for encryption and signature schemes," in *ACM SIGSAC*, pp. 399–410, 2013.

[20] D. R. Cok, A. Stump, and T. Weber, "The 2013 evaluation of SMT-COMP and SMT-LIB," *J. Autom. Reasoning*, vol. 55, no. 1, pp. 61–90, 2015.

[21] B. A. Nadel, "Representation selection for constraint satisfaction: A case study using n-queens," *IEEE Intelligent Systems*, vol. 5, no. 3, pp. 16–23, 1990.

[22] M. A. Salido and F. Barber, "How to classify hard and soft constraints in non-binary constraint satisfaction problems," in *Research and Development in Intelligent Systems XX*, pp. 213–226, Springer, 2004.

[23] V. Klebanov, P. Müller, N. Shankar, G. T. Leavens, V. Wüstholz, E. Alkassar, R. Arthan, D. Bronish, R. Chapman, E. Cohen, *et al.*, "The 1st verified software competition: Experience report," in *FM 2011: Formal Methods*, pp. 154–168, Springer, 2011.

[24] C. P. McCarty, "Queen squares," *American Mathematical Monthly*, pp. 578–580, 1978.

[25] D. A. Klarner, "Queen squares," *J. Recreational Math*, vol. 12, no. 3, pp. 177–178, 1979.