Distributed Resource Allocation and Management in Heterogeneous Networks

Dissertation

by

Gunnar Schomaker

Heinz Nixdorf Institute and Department of Computer Science
University of Paderborn
January 2009

Reviewers:

- Prof. Dr. Friedhelm Meyer auf der Heide, University of Paderborn
- Prof. Dr. Christian Schindelhauer, University of Freiburg

Contents

Chapter 1

Introduction

Don’t be afraid of the weight. It is resolvable and sometimes only attached!

1.1 Resource Networks

In nowadays, computers are very powerful and commonly very well equipped with resources like computational power and storage capacity for running applications and saving documents. Further, computers are more and more embedded within a network. Computers without a network device in use are rather the exception than the normal case. The general distinction of these networks can be made between local area network aka. LAN and wide area networks aka. WAN. As their name says, LANs are locally ranged, like home networks or company networks and the WAN is not locally ranged, but the organizing layer offers connections to other LANs.

Both networks consist of personal computers or dedicated units for special services like a printer, a mail server, file server or any other web service. Commonly connections to the WAN are established to access the world wide web or internet. Several hardware techniques are available to maintain these connections. A persistent connection ensures the availability of a computer or a whole network infrastructure. A major task within these networks is to search, access, store or exchange information and documents. Therefor, users connect directly to the service source offering the required resource. Independent of the kind of resource, there are two major problems if resources or services are offered for many others.

In the following, an example gives the first problem. If many users try to access in a network the same and very popular document or if many documents on a single source are requested, then the providing source, commonly a server, might become overloaded. The high number of requests will swamp the server and non of the users or only a dedicated set of requests can be satisfied. Eventually, because the necessary server bandwidth is a limited resource.

The second problem is the outcome of the solution of the first. To avoid swamped servers documents have to be distributed among several servers or a group of servers each offering the same content. The benefit of using multiple or/and redundant severs is clear, if a single source fails, the redundant one can carry over its tasks. If all sources are working properly, the sum of all requests can be balanced and distributed among all sources in such way that non of the servers gets possibly swamped. In contrast to spreading without redundancy, the availability of documents is increased if offered at many independent sources.

The disadvantage of redundancy or other codings is that each single source needs administration, consumes energy and finally money. The items or segments must be distributed among the recourses in advance. Further, each server should not handle more requests or documents he is capable of. Thus the number of access or allocation requests should reflect the servers expected fraction over

all requests. Furthermore, we have to ensure that there is always a spare server in charge if one fails. If all this is solved with respect to the heterogeneity of servers, we obtain an efficient resource management and an increased availability of the distributed documents.

Eventually two kinds of networks have to overcome these problems and are of interest for us. The first are *Peer to Peer Networks* aka. P2P Networks. From them we know techniques to distribute request among a dynamic set of peers efficiently, like CAN [37], Chord [48], Pastry [15], Tapestry [19], and many more. A description of Consistent Hashing can also be found in this work in Section 2.1.3. The second one are Storage Area Networks aka. SAN(s), because here we want to establish such techniques and adapt them for special needs of this application area. We will not discuss both networks in detail, but we will see and understand the differences subjecting allocation processes, heterogeneity, balancing and the impact of dynamics.

However, independent of P2P or SAN, each nodes participates with at least one specific resource, like bandwidth or storage capacity. If this resource is limited the sum of all resources is it too. For this we use the following definition

Definition 1. *Let V a set of nodes and $c_v = c(v) \mid v \in V$ the resources capacity of a node, then the network capacity is denoted by $C_V = c(V) = \sum_{v \in V} c(v)$.*

The objective of most approaches is to disburden a node as far as possible and doing this with respect to its capacity relative to other participating nodes. Thus, we have to know more about the available resources within the network and the share of a node in a network, defined in the following

Definition 2. *The relative **share** of $v \in V$ in a network is denoted and defined by*

$$s_v = \frac{c_v}{\sum_{u \in V} c_u} = \frac{c_v}{C_V} \tag{1.1}$$

So if the network is of size $|V| = n$ and the share of each node is $s_v = 1/n$ and its capacity is C_V/n, we know that all nodes are equal.

Mainly this work considers the special needs of Storage Networks, because in contrast to requirements of Peer to Peer Networks the storage allocation in SANs has to deal with a strictly limited resource. Thus, it is of tremendous importance that non of the nodes receive more than its share defines.

1.1.1 Storage Area Networks

The basic idea of SANs is to avoid the local resource concentration of single storage servers. The aim is to achieve server independent storage connected via dedicated or shared network. Many details considering the architecture of SANs,

currently used communication protocols, and available physical networks can be found in several books [50] [20]. In this section, we will introduce the main parts of a SAN, which are storage devices, and the most common distribution or allocation strategies. We do not consider the connecting network here and will not consider the network in the following chapters, because the main focus of this work are the distribution functions and their specific properties.

The Storage Disk Model

Hence persistent memory of both networks mainly consists of storage disks, we briefly introduce and discuss their evolution here. Today, magnetic disk drives (HDD) mainly consist of a controller, which processes requests and data is stored on a rotating magnetic platter. For accessing data, an arm moves a read/write head over a specific track on a platter. This phase is often described as *seek* and *settle* time. If, after a short *waiting* time above the track, the appropriate sector appears under the head, the data can be *read* or *written*. From this a rough estimation on the time to access a single data block b can be described by the summation of the following functions

$$f_{access}(b) = [seek(b) + settle(b)] + [wait(b) + operation_{r/w}(b)]$$

This clarifies why the access time of data patterns for consecutive blocks on a track is faster compared to widely spread patterns. For more details on modeling and access optimization on magnetic disk drives we refer to an other article [38]. Since each mechanical action generates non negligible latencies, a request cache is commonly embedded on a disk controller, and used for frequently accessed data. Data stored in this cache can be accessed immediately and causes no mechanical latency.

The current evolution of storage disks is influenced by the availability of huge, persistent and cheep flash memory. In contrast to former disk caches, the amount of used flash memory covers a significant fraction of available disk capacity. Further, the cached data is still available after power-down and can be used to accelerate the system booting or application start. Such a technique to include Flash Ram is successfully used by Microsoft's *SuperFetch, ReadyBoost, ReadyDrive*, which are included in Windows Vista and accelerate the operating system in several areas, and often referred as Hybrid Hard Drive Support [32]. Further, storing frequently accessed data on the flash memory of *Hybrid Hard Drives* (HHD) will reduce the access latency, less energy consumption and heat generation. This effect on energy consumption in a RAID system was researched by Frank Wang, et al. [52].

In future, mechanical parts of storage disks may be completely replaced by flash ram, thus some currently used models and their parameters will be obsolete. Nevertheless, the main disadvantage of *Solid State Drives* (SSD) is, that after mechanical failures data is completely lost as the cell is destroyed. While, if a normal HDD suffers by mechanical failures the data is often recoverable using expert help. Eventually, this might be at least one argument why HDD will not be extinct in the near future.

RAID

Current SANs normally consist of hard drives, a limited and restricted resource. In such a network a storage node does not decide whether it wants to store data, like peers or users in a P2P Network do. Likewise, an allocation request, compared to a read request is much more persistent. If a fraction of resource is once allocated, it will reduce the free capacity unless it is deleted. Thus the used allocation or distribution strategy for items should waste nothing and should not allocate more than available on a device.

A classical and major data distribution technique in SANs is described in the RAID level scheme [34]. Its original abbreviation was *Redundant Array of Inexpensive Disks* and developed to compensate the cost for huge and fast disks with smaller once using parallelism. The nowadays meaning of RAID is *Redundant Array of Independent Disks*. Over time aspects like fault-tolerance have been included. It is derived by combining data segmentation aka. *striping* with data duplication aka. *mirroring*, data coding like *XOR parity*, or improved erasure codes. However, the name change indicates the evolution of its current focus, independency of devices. Essentially, all *standard* or *nested RAID Levels* are based upon this independency. The placement on distinct disks is of major importance, since mirroring is senseless, and striping does not improve the access speed. Especially in case of inaccessibility of disks coding might not be invertible and thus inoperable if a failure occurs. At least the static distribution function of RAID grants an even distribution of items among all disks. This implies that the system is balanced according to capacity usage, unless the configuration of the SAN does not change.

The distribution unit within RAID levels are typically blocks. Each physical address of a data block is determinable by a simple function, depending on the specific RAID Level. Let $V = \{v_0, \ldots, v_{N-1}\}$ a set of N equal devices, where each $v \in V$ has 2^i allocable blocks, then we have $N \cdot 2^i$ addresses for blocks. Basically, the subsequent allocation of blocks on alternating disks is determined by a modulo function using the gathered address room and p, denoting the redundancy or striping defined by the RAID Level. Details on this can also be found

in many books [50]. However, the most disadvantages is that changes in V will destroy this strict mapping order, leading to costly data movements to preserve data consistency.

In the following we need a flexible notation defining such access patterns among V. So we denote with $\bar{v}$ an access pattern which addresses or allocates storage among p not necessarily distinct disks.

Definition 3. *Let V a set of nodes, then $\bar{v} \in V^p$ is independent, if $\forall\, i, j \in \{1, \ldots, p\}$, with $i \neq j$ also $\bar{v}_i \neq \bar{v}_j$ is true. Further, $\bar{v}$ can allocate at most $c(\bar{v}) = p \cdot min(\{c(\bar{v}_i) \mid 1 \leq i \leq p\})$.*

The definition of RAID implies its need of a set of independent patterns to operate reliable. In case of $c(v) = c(u)$ for all $v, u \in V$ and $|V|/p \in \mathbb{N}$ such patterns are easily constructed. Therefor, we have to group V into p subsets $V_1, \ldots, V_p$, such that for all $V_i \cap V_j = \emptyset$ if $i \neq j$, where each set consists of $\lfloor N/p \rfloor$ consecutively numbered devices. This we can do by using the mod operator on integers if V is indexed like $V = \{v_1, \ldots, v_{|V|}\}$. Thus, a suitable set of independent patterns is defined by $\bar{V} := \bigcup_{1 \leq i \leq |V|} \{\bar{v} := (v_{(i \bmod p),i}, \ldots, v_{p,i})\}$. By this, we guarantee that each device occurs only once over all patterns. Note, if $|V|/p \notin \mathbb{N}$, then at most $p - 1$ devices are left unused.

RAID and Heterogeneity

In SANs many strategies are known to distribute content among several disks, but less are able to respect different disk capacities or bandwidths and only a few include dynamics if the system configuration changes.

Typically, a block device system utilize n disks by defining a virtual block of size $|b_v|$, describing the data fraction each disk derives or has to deliver if requested. The original strategy was to divide such blocks in same sized pieces of $|b_v|\frac{1}{m}$, where $\frac{n}{c} = m \in \mathbb{N}$. Thus, using disks with same capacities is reasonable if no capacity should be left unused.

Several aspects of data partitioning and load balancing in parallel disk systems are discussed in Scheuermann et al. [41]. They have discussed striping including technical aspects like head positioning and arm movement with respect to average request size over all files. Unfortunately they considered less devices compared to nowadays, where a SAN system contains tens or hundreds of disks. Nevertheless, many of their observations are still important if parallelism is an issue.

Cortes et al. [13] presented a static solution to overcome the problem of uniform disks by introducing *AdapRraid0*. They used virtual blocks where the fraction of each disk reflects its capacity ratio compared to each disk included in such

block. Nevertheless, they assumed that disks with higher capacities are faster and thus the result improves the bandwidth. Zimmermann et al. [54] have used a similar approach, but they focussed in *HERA* on reliability and presented a scheme that includes redundancy based on parity information and presented an analytical Marcow model of reliability in heterogeneous disk arrays.

If such systems using virtual blocks are changed by replacing a disk d with a new disk d' and $c(d) < c(d')$ the additional capacity $c(d') - c(d)$ is left unused or intensive recalculations must be done to preserve consistency. Similar effects happen if disks are added or removed. Furthermore the size of a virtual block grows linear in the number of disks it joins, on the one hand this leads to wasted storage if files are small and on the other hand if the block size is constant the benefit of parallelism decreases caused by the seek time on each disk.

Di Marco et al. presented a distributed disk array architecture called *DRAID* [27] accessing $N + K$ disks in parallel. They used a gigabit Ethernet cluster infrastructure and improved the fault tolerance compared to RAID IV. They included Reed-Solomon error correction organized in a two dimensional matrix. They argued that the need for such multi fault tolerance is justified by the average uptime of systems in such an infrastructure. They have considered scaling of their system too, but eventually adding single disks is inoperative, because of the recalculation needed for the Reed-Solomon encoding [35]. To overcome this, they suggested to add disks in multiples of $N + K$.

Hence the storage consumption is ongoing increasing, one very important issue in nowadays is scaling or adaptability. Since the available capacity of devices is still growing, changes should respect the possible device heterogeneity in a single SAN. Furthermore, in SANs it is of most importance to support the already established RAID Levels without cut backs. As seen, all mentioned schemes have restrictions or problems if disks are added or removed from the environment. Especially, if disks are different in capacity and/or bandwidth. Classical approaches can not react efficiently and preserve data consistency, concerning data movement. Brinkmann et al. [10] tried to overcome this and introduced a scheme using consistent hashing to resolve heterogeneity for SAN purposes. More about this method we will see in Chapter 2. An additional challenge of these newer techniques is to achieve a balanced allocation and coevally allow to distribute multiple instances of the same item among distinct devices. Brinkmann et. al [8] and in an other work Mense et. al [28] have faced this problem. Both presented solutions are also capable to support all those RAID Levels which rely on a distinct node selection.

1.2 Our Contributions and Results

The fundamentals of this work are derived by ideas first discussed in a student project group, implementing an own Mobile Ad Hoc Network called PAMANET [5] [42]. The PAMANET consists of three main components. First, the embedding of the routing layer into IEEE 802.11 and IPv6 by using techniques used at the ad hoc support library (aslib) by Gupta et al. Second, the routing layer which combines a landmark routing, hierarchical clustering, consistent hashing for providing location dependent addresses and lookup-service for the location of nodes. Third, a Peer-to-Peer data storage system based on egoistic distributed caches enabling hop and traffic efficient data access on replicated data partitions.

From this work we have enhanced the second key technology, called Weighted Consistent Hashing or later Distributed Heterogeneous Hash Tables, aka DHHT, introduced at SPAA 2005 [43]. It consists of two methods applying weighted distributed hash tables to resolve the heterogeneity of nodes. The first method, called Linear Method, combines the standard consistent hashing introduced by Karger et al. [21] with a linear weighted distance measure. By using a constant number of node copies and distinct partitions within the hash space, the balance of this scheme approximates the fair weight relationship with high probability. The second method, called the Logarithmic Method, uses a logarithmic weighted distance between the nodes and the data to find the corresponding node. For distributing one data element it provides perfect weighted balance if $\mathcal{O}(\log n)$ partitions are used. Both methods provide small fragmentation, which means that the hash space is divided into at most $\mathcal{O}(n \log n)$ intervals. Furthermore, there is an efficient data structure that assigns data elements to the nodes in expected time $\mathcal{O}(\log n)$. If fragmentation is of no issue, one can replace the use of partitions by a further method we call Double Hashing. This method needs O(n) steps for assigning an element among a set of nodes. It is recommended for small node sets, because it overcomes the negative side effects of fixed hash range segmentations. However, all methods can be used for Storage Area Networks, where the number of nodes is small compared to Peer-to-Peer networks and where global knowledge about nodes is typically available. Details of this wok can be found in Chapter 2.

Further, we have considered content delivery networks for large files, like movies [24], [25]. There, we have combined dynamic data replication with different capabilities of content delivery nodes. The use of redundancy, was inspired by early results [6]. The replication was controlled by different access frequencies on data, forecasted by an ARIMA time series model. The aim was to reduce the delivery latency by occupying unused storage resources with replicates and fairly share the bandwidth of content nodes. Details on ARIMA time series model, data

structures and algorithms can be found in an other dissertation [23]. Moreover, the recent article in this topic was decorated with a best paper award [25].

Inspired by the advantage of parallelism we have described an efficient solution for providing parallel access to multiple hard disks for popular content [44]. In extension to previous approaches we have provided an efficient hash table data structure for utilizing the full capacity of each data server. Concerning the dynamics, documents as well as servers may be added or removed from the system causing only local changes. We have considered sequential and parallel access to data in the average case and for the average time model we presented an optimal algorithm.

The idea of using parallelism in combination with load balancing and DHHT was first considered and used in web computing [16]. There, we have presented an architecture for distributed computing in a Peer-to-Peer network. In particular, we realized the Paderborn University BSP-based Web Computing Library (PUBWCL [7]), which formerly used a centralized client-server architecture for scheduling and load balancing, as a pure Peer-to-Peer system. By using DHHT, the architecture was capable to feature scheduling and load balancing of tightly coupled, massively parallel algorithms in the bulk-synchronous (BSP [51]) style with a minimal number of migrations. Furthermore, the architecture was capable of heterogeneous BSP programs, whereas the former version of PUBWCL could only handle homogeneous BSP programs.

These extensions, allowing parallelism are able to break the consecutiveness of an extent and allow parallel access on distinct nodes, see Section 2.1.1. Without considerable effort, this was previously possible by using the results of a patent we made [47]. Furthermore, load balancing and self administration by using DHHT, finally adapted for SAN purposes was introduced as DHHT-RAID [45]. The main topic of this work was a scaleable architecture capable to serve all RAID Levels at any time coevally. The DHHT-RAID architecture is able to store its meta information in its own structure, allowing to react on dynamical environment changes without the need of central management structures. Details and some simulation results showing the performance of DHHT-RAID can be found in Chapter 4. Additionally, we have made some simulations showing the impact of decomposing the hash range for nodes deterministically within DHHT. The random node position sampling was substituted by a greedy algorithm providing a constant smoothness between $\approx 1.118 \leq s \leq \approx 1.809$ and a linear hash range fragmentation with a constant number of different interval lengths. Details on the deterministic decomposition can be found in Chapter 3 [46].

1.3 Tools

Several strategies we are referring to and some approaches we introduce are randomized algorithms. For their analysis we need some tools and apply some abstract concepts which are motivated by theory. These tools and concepts are the following.

Definition 4. *Let $X(X_1, \ldots, X_n)$ be a function of n variables, then we say X is bounded by $\mathcal{O}(f(n))$ with* **constant probability** *(abbriviated with* **w.c.p.***), if for any ϵ, there exist a c such that*

$$Pr[X \leq c \cdot f(n)] = Pr[X \geq \mathcal{O}(f(n))] \leq \epsilon$$

Definition 5. *Let $X(X_1, \ldots, X_n)$ be a function of n variables, then we say X is bounded by $\mathcal{O}(f(n))$ with* **high probability** *(abbreviated with* **w.h.p.***), if for any $\alpha \geq 1$, there exists a corresponding c such that*

$$Pr[X \geq c \cdot f(n)] = Pr[X \geq \mathcal{O}(f(n))] \leq \frac{1}{n^{\alpha}}$$

In the following we show a technique bounding the probability that a number of independent random variables will deviate from their expectation values, called Cherhoff Bounds [40]. These bounds are commonly used to obtain sharper bounds on such tail probabilities.

Theorem 1. *Let $\{X_1, \ldots, X_n\}$ a set of n independent random variables and $X = \sum_{i=1}^{n} X_i$, where $\mu = E[X]$ denotes the expectation of X, then for all $\delta \geq 0$ the following holds,*

$$Pr[X \geq (1+\delta) \cdot \mu] \leq e^{-\min\{\delta^2, \delta\} \cdot \mu}$$

moreover, if it holds that $0 \leq \delta \leq 1$, than

$$Pr[X \geq (1-\delta) \cdot \mu] \leq e^{-\delta^2 \cdot \mu / 2}$$

Davenport Schinzel Sequences have some geometric applications. For us they are important, because they allow to derive method attributes subjecting the hash range fragmentation. Therefore we consider the lower envelope of a set of functions.

Definition 6. *Let $F = \{f_1, \ldots, f_n\}$ be a collection of n real-value, continuous totally defined functions, than the lower envelope of F is defined as*

$$E_F(x) := \min(f_i(x) \mid f_i \in F),$$

so E_F is the point wise minimum of the functions f_i.

The formal definition of a Davenport Schinzel sequence, aka. $DS(n,s)$-sequence is the following

Definition 7. *Let n, s, m positive integers, a sequence $S = \langle s_1, \ldots, s_m \rangle$ is Davenport Schinzel Sequenze if the following conditions are true:*

1. $1 \leq s_i \leq n$ *for each* $i \leq m$
2. $s_i \neq s_{i+1}$ *for each* $i < m$, *and*
3. *there are no* $s+2$ *indices* $1 \leq i_1 < i_2 < \cdots < i_{s+2} \leq m$ *such that*

$$s_{i_1} = s_{i_3} = s_{i_5} = \cdots = a, s_{i_2} = s_{i_4} = s_{i_6} = \cdots = b$$

and $a \neq b$.

The third conditions forbids long alternations of any pair of distinct symbols in the sequence. Interesting is that many problems can be reduced by a geometric interpretation of $DS(s,n)-$sequences. Especially the lower envelope of a set of univariate functions. The resulting bounds on the length $\lambda_s(n)$ of such a sequence or envelope, where the sequence S is given by the indices of the functions, can be attained by attributes of $DS(s,n)-$sequences. Details on this can be found in the a survey [2] or accordingly in the authors book [1].

In this work most of the randomized algorithms use an unit range $M = [0,1[$, where some of them interpret the range as an unit ring. Therefor, we will sometimes use the following re-definition of the modular function on real values.

Definition 8. *We redefine the* mod *function as*

$$a \bmod 1 := a - \lfloor a \rfloor$$

Further, most methods in the following are using hash functions to imitated a random like mapping into the hash range M. In the Following some assumptions and attributes we make on these functions.

Definition 9. *A hash function $h : M \mapsto N$ is said to be perfect for a set $X \subseteq M$, if for any distinct $u, v \in X$, $h(u) \neq h(v)$.*

Definition 10. *A family of hash functions $H \subseteq \{g \mid g : M \mapsto N\}$, is said to be k-universal with $k, l \in \mathbb{N}$ $H = h : M \mapsto N$, if for all $l \leq k$ and pairwise distinct $m_1, \ldots, m_l \in M$ and all $n_1, \ldots, n_l \in N$*

$$Pr[h(m_i) = n_i \mid \forall i \in \{1, \ldots, k\}] \leq \left(\frac{1}{|N|}\right)^l \tag{1.2}$$

The aim of using a unit has range M in combination with hash functions behaving like independent random variables is to transform the continuous problem into a discrete balls into bin model. This is done by random sampling of positions in M from which we derive the size of a bin. But, even in the case that the bins are all of same size we can observe an imbalance during the experiment. This can be explained by the a classical problem in random theory, the *Coupon Collector Effect*. Results and analysis on this classical problem can be found in several books [31]. It considers n types of coupons and m trials of choosing one of the coupons. The random choice of any coupon is mutually independent and equally likely to be chosen. The part of interest is the relation between n and m, if we ask how many trials are needed to collect at least one copy of each type. Actually all of our presented algorithms reflect this effect, too. This can be explained and reconstructed, if we define each node as a coupon type. Further, the hash function which imitates the random selection of nodes by distributing items among them.

Chapter 2

Distributed Resource Allocation

This chapter starts with the problem introduction and a general view on specific problem properties for an item distribution in a network among nodes with limited resources. We will discuss two specific problem variations, distinguishing each other by their node set composition. For both variants we present general models and former strategies, like Consistent Hashing, Share, and Sieve. We continue with the presentation of a our DHHT approach, which offers for both problem variants a solution. Since we consider our approach as a generalization and heterogeneous extension of DHTs, we show the equality of our model in the uniform case. Afterwards we continue with the transformation to the heterogeneous problem, on which we focus from then. Further, we show general properties of our approach consisting of two distinct methods, the Linear Method and the Logarithmic Method. As a consequence of the results and properties we enhance the requirements of the general model. In an additional section we consider special features of our methods, applicable for many areas, where balanced content distribution with limited resources is an issue.

2.1 The Distribution Problem

In this section we regard solutions for distributing a set of items among a set of servers or peers, in the following denoted as nodes. Coevally, we try to respect certain properties of these nodes to achieve our distribution objectives. The general model consists of three scopes, the item set D we distribute, the node set V storing the items, and finally the distribution or mapping function f to determine the mapping of items to nodes. Altogether we denote from now on as the environment (V, D, f).

In the following the detailed environment description. $V := \{v_1, \ldots, v_n\}$ defines the node set which is either *dynamically* or *statically*. This means, in the dynamic case nodes may join or leave V over time. In contrast the static case, where the number of nodes is predefined and fixed. Furthermore, there is a set of weighting functions $W := \{w_i : V \to \mathbb{R}\}$ each assigning weights $w_{i,v} = w_i(v) \in \mathbb{R}$ to all $v \in V$. For every node v, according to the attribute i, the function $w_i(v)$ determines a weight.

So, this dissimilarity of nodes or heterogeneity within V is reflected by such possibly unequal weights. This leads to a further notation, if for a fixed i and $\forall$ $v, u \in V$ $w_{i,v} = w_{i,u}$ is valid we say V is *uniform in* w_i, but if there exists a any pair (u, v) with $w_{i,v} \neq w_{i,u}$ we say V is *heterogeneous in* w_i. If we consider only one attribute and thus $|W| = 1$ we say V is either *uniform* or *heterogeneous* and we use the abbreviated notation w_v.

Beyond nodes, $D = \{d_1, \ldots, d_m\}$ defines the set of items we want to distribute among V. Analogous to V, we say D is either a *dynamic* or a *static* set. Additionally, we define a set of weighting functions $S := \{s_j : D \to \mathbb{R}\}$. Again, each function assigns an appropriate weight $s_{j,d} = s_j(d) \in \mathbb{R}$ to every item d and its attribute j. Same as before for nodes, the notation regarding the heterogeneity of D or if $|S| = 1$. Further, we will say an environment $(V, D, f_{W,S})$ is dynamic or uniform if an attribute is true for V and D. The central and assessable environment part is the mapping function $f_{W,S}(d) : D \to V$. It assigns items to nodes, with respect to W or S and further defined characteristics of V and D. Finally we denote with $f^{-1} : V \to D$, where $f_V^{-1}(v) \subseteq D$ the function that determines the stored items on a node $v \in V$.

Summarized, we describe the mapping of items among nodes within a certain environment $(V, D, f_{W,S})$. In our model we only consider nodes and items, we do not take care about the limits of the connecting network. Our main challenge is to find an appropriate mapping function, which achieves the desired distribution objectives and copes the demands of its application area efficiently.

For clarity and to exemplify the model let us consider some application areas. In Storage Networks w_i possibly ascertains the allocable storage capacity,

the maximal available bandwidth or other attributes of interest of each storage node. In some cases it might be reasonable to combine node attributes. For instance, a combination of capacity and bandwidth, based upon the assumption that the average bandwidth/ throughput increases with the node capacity. Nevertheless, such a combination must be done very carefully! For example, let $w_{1,v}$ describes how fast v can receive, $w_{2,v}$ how fast v can deliver and $w_{3,v} = c(v)$ how many items v can store. Applied for Magnetic Disk Drives (HDD) $w_{1,v}$ and $w_{2,v}$ depend on the current total number of competing parallel accesses and their distribution between $w_{1,v}$ and $w_{2,v}$. In contrast, the current access distribution behavior has no influence on $w_{3,v}$. So an explicit independent description of each node attribute seems often to be more helpful to handle such dependancies and to identify whether they are important for the distribution objective or not.

One possible and reasonable distribution objective is an even capacity allocation over all nodes in V. So let $s_{1,d}$ denote the size of an item, then the used storage capacity of a node v should be

$$\sum_{d \in f_V^{-1}(v)} s_1(d) \approx \frac{w_3(v)}{\sum_{u \in V} w_3(u)} \cdot \sum_{d \in \cup_{u \in V} f_V^{-1}(u)} s_1(d) \leq c(v) \tag{2.1}$$

It is of much importance that the distribution function can provide this at any time and as close as possible, with respect to the upper right constraint. This means, if the distribution function has to handle a dynamical environment, it has to compensate allocation deviations by leaving or joining nodes. Further it has to ensure that each node has to store only as much items as its capacity allows, especially during the maintenance operations. This is an issue, because items stored on leaving nodes should not get lost. To avoid this, a protocol based on the distribution function or method needs to replace items in advance. As we have seen in Section 1.1.1 such changes in a storage environment can be fixed for some distribution methods only inefficiently and we will see later how this can be resolved. If a similar objective should be presumed for the item request distribution, one has to differentiate whether items are distinct in their access frequency or not. Independent on reading or changing the item, if not, it is the same we have in Equation 2.1. If yes, the objective is not that simple and clear any more and can become very complex. A reason therefor is the specific arrangement of node and item attributes. An even item distribution might contradict the aim to minimize the delivery time of items, and a distribution optimization for delivery time might exceed the capacity bound of equation 2.1.

In application areas like Peer-to-Peer Networks, where the transient participation of nodes aka. peers is the general case, the focus lies on the redistribution of traffic on content. In particular, if items are very popular, this is done more or

less automatically. Hence, peers are coevally sharing the content they are downloading, popularity gets automatically resolved by the redundancy of multiple accesses, which results into a higher item availability. Since peers are normally leaving the network if they have accomplished their download, it is important for the previously connected nodes to find appropriate replacements to continue the download with the same pace. So a fast and robust lookup structure to identify adequate node replacements is one main target here.

Beside the specific behavior of nodes and their abilities, items are, beneath their content, different too. They differ in the amount of memory they allocate on a single node or their popularity. Some items are more and others are less popular and the popularity changes permanently. In Peer-to-Peer like networks the popularity results into a varying number of redundant locations at specific time points and the sum over all locations leads to a varying memory consumption in the whole network. So if the access frequency for an item is high, it implies more bandwidth consumption for the offering node, too. A possible strategy to avoid node thrashing by this, and coevally decreases the delivering time for items is an active creation and placement of replicates on additional nodes to discharge them. Especially, nodes with less popular content and enough free storage capacity and bandwidth are good candidates. Actually, such an active replication is nothing else than provider do if their web servers are offering popular sites. They simply dynamically distribute requests among multiple web servers with identical content. This is an approved strategy, if the aim is to obtain an evenly spread workload over all servers during prime times [21]. Notice, the number of requests a server handles is not exactly the same and they are read only. This is sufficient, if the magnitude of request is the same and finally such read requests are not persistent. In contrast to this, if the same function is used to determine and allocate item places, it possibly violates the restriction of Equation 2.1.

Heterogeneity of items combined with node abilities is not the general approach, but if, nodes should be classified. Classification attributes might be the number of items a nodes is responsible for, the up- and downstream bandwidth, the maximal number of parallel connections from other peers or the connectivity degree within the overlay network, and so on. If the organizing overlay network offers an order respecting the heterogeneity of peers [4], such an embedded order can enhance the major task of the network. As mentioned before, it sometimes makes sense to combine attributes like the number of parallel accesses and the available node bandwidth. If provided, accessing peers are able to estimate the minimal bandwidth a peer can offer for a specific download. If a peer with a high bandwidth offers many popular items and allows unlimited connections, the bandwidth per connection might possibly decrease dramatically. So, to improve the termination time, it could be much wiser for a peer to connect to a

slower node which offers less other popular items or less parallel connections. Loeser et. al [25] have used this and embedded a request forecasting system to control the replication rate of video files. The aim was to increase the average system output by decreasing the delivering time for popular files, if popularity of files and capabilities of serves like capacity and bandwidth gets combined.

Within Ad-hoc Networks the size or the responsibility of a peers may grows simultaneous with the time a peer stays within the Network [5]. So, the value is continuously increasing over time. This kind of weighting is based upon the general assumption that the probability for a peer to leave the system decreases the longer it stays within the network. Eventually, this assumption helps to optimize routing purposes.

Finally, to realize the distribution task, and to fulfill the objectives of the distribution function can be very difficult. Especially, if the demands and constraints hardly depend on the application area. Further, since $f_{W,S}$ determines the location of an item, already mapped items must be relocatable after changes in V occur. So, the number of steps needed to repair the system and obtain the distribution objectives again is indirectly controlled by $f_{W,S}$ too. This aspect requires special consideration in the design of an efficient mapping function. Eminently, if there are costs for every repair step, preserving the mapping consistency, this is not negligible. A further aspect of particular importance in the design process is, how much information about the current state of $(V, D, f_{W,S})$ is used by $f_{W,S}$ to determine the mapping or how many repair operations are needed to fulfill these predefined requirements. The necessity for global information, like the current item distribution over all nodes, to apply operations efficiently is given by $f_{W,S}$. Especially in the area of Peer to Peer Networks the efficient communication is of much importance. Nodes decide and act only based upon local information, which nevertheless should result in a good global behavior.

In this work we will focus on the description and analysis of mapping functions for items to nodes. We set the following requirements for the environment $(V, D, f_{W,S})$. Since we are strongly related to Storage Area Networks, we consider V as a dynamical set of nodes, where each node offers a limited storage capacity for placing items. We normally assume that nodes can differentiate in capacity, so $|W| = 1$ and for simplification use $w_{1,v} = w_v = w(v)$. It is important to keep in mind that the storage capacity on a node is a limited resource and placed items are representing persistent requests on each node. However, the item set D is dynamically too, but because of random like item mapping we will mainly discuss the insertion of a single item. Further, exempt from content, items are equal, so D is homogeneous and $|S| = 1$, analog the simplified notation. As seen before, this restriction is somehow artificial, but the diversity of items is not discussed here. For further reading and comparison with other approaches we refer to another

work [23].

2.1.1 Formal Definition

As already motivated the mapping function objectives can be very different. To allow the comparison and quality measurement of mapping functions or algorithms we give the following definitions.

Definition 11. *The mapping function f is marked as **space balanced** in an environment (V, D, f), if for each $D' \subseteq D$, any node $v \in V$ gets at most*

$$M_V(v) = \left\lceil |D'| \cdot \frac{w_v}{\sum_{u \in V} w_u} \right\rceil = \left\lceil |D'| \cdot \frac{c_v}{\sum_{u \in V} c_u} \right\rceil = \lceil |D'| \cdot s_v \rceil \tag{2.2}$$

items. Especially for distribution strategies based on an item mapping using a random distribution process, each $v \in V$ should get at most

$$M_V(v)' := M_V(v) \cdot (1 + \epsilon) \tag{2.3}$$

items.

For random distribution strategies this equation should hold for any constant $\epsilon > 0$ and optimally very close to zero. The Definition 11 relies on the fact that D is considered as a homogeneous set. It can be derived by the simplification of Equation 2.1, if items are considered as inseparable units. Thus $s_1(d)$ can be substituted by 1, which i a valid assumption, since we say that each file can be represented by multiple units of constant size. In storage devices this is typically a *Block* or in lager system or networks an *Extent*, which is nothing else than a fixed grouped number of blocks. The arrangement of blocks within extents can be consecutive in the physical address room of a single device. Likewise, it can be determined by an arbitrary bit mask, identifying the address bits of a single device among all devices [47]. Such a bit mask typically breaks the allocation consecutiveness of an extent on a single device.

Beneath space balance of nodes, we further consider dynamical changes in the environment (V, D, f) and its impact. Especially changes in V are of much interest. As mentioned, f is indirectly responsible for the number of steps needed to keep the availability of items during and after such operations. Therefore, a distribution strategy without using replication has to react in advance to keep space balance and thus to guarantee availability of items coevally. The efficiency of necessary movements in advance to satisfy this aim are considered by the following definitions.

Definition 12. *We say a distribution strategy is **adaptive** in the dynamical environment (V, D, f), if it is able to achieve space balance after changes in V or D.*

So, the objective of space balance combined with the need of availability leads to adaptiveness. Since the property of adaptiveness can be achieved in many ways, including inefficient once, we need a further quality measure. It is reasonable to assume that the number of steps needed to achieve space balance should be low. We assume that before the operation occurs (V, D, f) was space balanced. For a leaving node v this implies to evacuate each item stored on it. In the following we denote with a step such an evacuation of a single item. This implies for the optimal number of steps if a node leaves

$$leave_{OPT}(v) = |f^{-1}(v)| \leq \left\lceil |D| \cdot \frac{w(v)}{\sum_{u \in V} w(u)} \right\rceil \leq \lceil |D| \cdot s_v \rceil \tag{2.4}$$

and the optimal number of steps for a node v joining the system should be the number of item it needs to store to achieve space balance this implies for the optimal number of steps

$$\begin{aligned} join_{OPT}(v) &\leq \left\lceil \sum_{u \in V} |f^{-1}(u)| \cdot \frac{w_v}{\sum_{u \in V} w_u} \right\rceil && (2.5) \\ &= \left\lceil \sum_{u \in V \setminus \{v\}} |f^{-1}(u)| \cdot s_v \right\rceil \leq \lceil |D| \cdot s_v \rceil && (2.6) \end{aligned}$$

Note, in Equation 2.5 we have presumed that joining nodes are empty and receive their content after the insertion. However, from both equations we can see that the optimal number of steps does not depend on the occurring operation, but on the leaving or joining node, its capacity and the current system capacity load. To compare the efficiency of adaptive strategies we count the number of steps needed to achieve space balance and compare them with the minimal number necessary.

Definition 13. *We say an adaptive distribution strategy f in a dynamic environment (V, D, f) is **c-competitive**, if the needed number of item movements, we count as steps, to achieve space balance is*

$$c \cdot steps_{OPT} = steps_f \geq |D'|, \tag{2.7}$$

where $D' \subseteq D$ denotes the items moved, $setps_f$ the number of used steps and $steps_{OPT}$ the optimal number of steps needed.

So the best a strategy can reach is 1-competitiveness or an $(1+\epsilon)$-competitive ratio in the random case. Beneath the importance to differentiate whether the space balance is obtained directly or if further movements were necessary to reach the final item distribution, we also want to attest a strategy specific data flow characteristic. Therefore, we look if an item lands directly on its final destination node and if dynamic interactions between other nodes were needed to keep the system balanced, too.

Definition 14. *Let (V, D, f) be a dynamic environment and $D' \subseteq D$ the set of items moved after a join or before a leave operation of node v. We say that a distribution strategy f is **monoton** for join or leave opetations if elements in D' move only from or to v.*

So this definition gives us the information whether items are migrating only in cooperation with the joining or leaving node or also among other nodes.

2.1.2 The Uniform Case

In this section we consider solutions subjecting the Uniform Case. The compliancy to the definitions of Section 2.1.1 are important, if we want to evaluate solutions targeting a specific case. In the environment, pictured in Figure 2.1, we assume that V is dynamical and uniform and we consider only the storage capacity of nodes. Further, we assume that D is dynamical and items uniform.

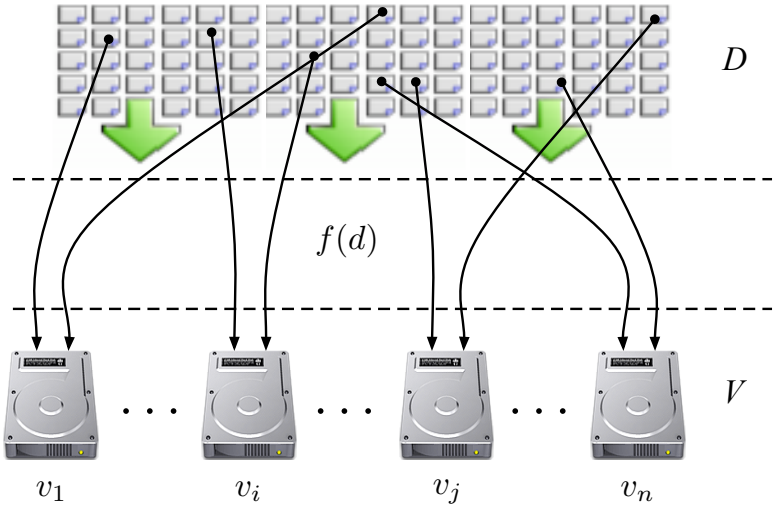


Figure 2.1: The Uniform Case

Each item allocates the same amount of resources on a node, so we can apply the definitions of Section 2.1.1. The targeted properties and requirements of an item mapping f are summarized as follows:

Definition 15. *Let (V, D, f) be a dynamical and uniform environment. A mapping function f solves the uniform case, if the following properties are fulfilled.*

1. *The mapping of $d \in D$ by f must be simple. This is the case if f uses as input V and d without the knowledge of $\bigcup_{v \in V} f^{-1}(v)$.*
2. *The mapping is space balanced or fair if for any $v, u \in V$ $f^{-1}(v) \approx f^{-1}(u)$*
3. *The mapping function f is adaptive in case of changes in $|V|$ and monotone for join and leave operations*

The requirement of Definition 15.1 for f implies that an item mapping should be independent of each other. So we should try to distribute an item without the knowledge of the current state of (V, D, f) or future mappings of $D \setminus \bigcup_{v \in V} f^{-1}(v)$. Further, Definition 15.3 implies that we want to preserve all these properties after changes in the environment have occurred.

2.1.3 The Uniform Case and some Solutions

In this section we describe methods solving the uniform distribution problem. Such solutions can be classified globally into two types, deterministic and randomized approaches. The most important deterministic approaches are discovered in the application area of SANs and are defined by several RAID Levels, see Section 1.1.1. In this chapter, we want focus on methods which determine a mapping by the use of random or random-like functions.

A Pure Random Model

Let us start to consider a simple random model for item distribution. Within an uniform and dynamical environment (V, D, f) f is implemented by the following mapping procedures:

1. $insertItem_D(d)$: chose any $v \in V$ randomly, where the probability of v to be chosen is $1/|V| = 1/n$.
2. $deleteNode_V(v)$: take all items $f^{-1}(v)$ and redistribute them among $V \setminus \{v\}$ using the method $insertItem_D(d)$.

3. $insertNode_V(v)$: ask any item $d \in \bigcup_{u \in V \setminus \{v\}} f^{-1}(u)$ if it wants to stay or leave by random, where the decision stay has a probability of $1 - 1/(n+1)$ and leave $1/(n+1)$.

Essentially, this mapping is equivalent to a Balls into Bins model with uniform bins and uniform items. Of most importance is that commonly $|V| << |D|$. Which is the case, if you consider magnetic disk drives as nodes and blocks or extents as items. Since $|D| \geq |V| \cdot \ln |V|$ bins receive with heigh probability balls in the order of the mean value. Nervertheless, there always exists a bin with

$$\frac{|D|}{|V|} + \Theta\left(\sqrt{\frac{|D| \cdot ln|V|}{|V|}}\right) \tag{2.8}$$

Since nodes with a fixed capacity can not store more balls than place is available, it is of much importance to avoid such heavy loaded bins. A common technique to achieve less over allocation is multiple choice. For further readings considering space-balance in balls into bins games with or without multiple choice, we just refer to the results in [36], [30, 29], [3], [18]. We can see that the algorithm $deleteNode_V(v)$ moves only balls from the existing bins to the new one and analogous, but in the other direction, for the $insertNode_V(v)$ algorithm. A problematically behavior is embedded in $insertNode_V(v)$, where the decision is made whether a ball moves to the new bin or not. Therefore, the algorithm asks every item whether it wants to stay or leave. Actually, this is the same as if the whole experiment is repeated with $n+1$ bins. So, asking violates the condition 1 of Definition 15. The only advantage of asking instead of reiterating is the avoidance of additional item movements. Additionally, touching each item separately in an distributed environment and asking independently in Bernoulli experiments whether it wants to stay or leave means a lot of communication.

Corollary 1. *The simple random model is adaptive, $\mathcal{O}(1)$ competitive in expectation and monotone for leave and join operations. It does not solve the uniform problem completely, because it violates Definition 15.1*

Beside the positive attitudes of a simple random Balls into Bins games we have to point out another disadvantage of interest. A distribution strategy for item allocation requires in contrast to a distribution strategy for transient requests a look up mechanism. Though, if items are thrown randomly using a random number generator implies that the same item can possibly be assigned to every bin. So the best one can do to overcome this, is to create an additional data lookup structure and to store each record consisting of the item id and the assigned bin indices. This kind of meta information uses space and if a meta data record is

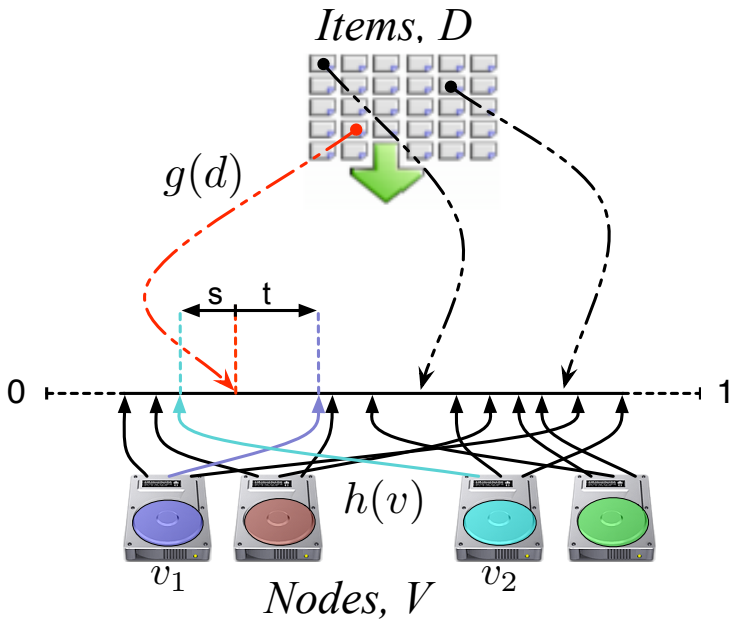


Figure 2.2: The Mapping of Consistent Hashing, including Virtual Nodes

of same size as the items we distribute, we have won nothing. The system we have built consumes the same amount of data for managing as it offers for item. So to do this more efficiently the item size should be much larger than the meta data record. Further, to get a unique ID for each item one uses typically hash functions $h : X \to [0,1[$. We will not discuss the properties of these functions here. However, the following approaches assume, that the hash functions behave like independently distributed random variables. Thus the expectation for an element in X to be mapped to a specific value in the range of $[0,1[$ is for all values the same, see Section 1.3.

Consistent Hashing or Distributed Hash Tables

As we have ascertained in the previous section, for an efficient item mapping covering the requirements of Definition 15 we have to consider more than balancing and adaptivity. An approach that gets rid of the additional and external lookup structure and keeps the positive characteristics of random experiments, was introduced by Karger et al. in [21]. It is a fundamental scheme called *Consistent Hashing* aka. *Distributes Hash Tables* often abbreviated with *DHT* and uses a two-sided ranged hashing scheme for nodes and items, into some continuous range

$M := [0, 1[$. In the original setting data was to be distributed among different sets of hosts, so called views. The goal was to avoid swamped servers, coevally decrease the usage of memory and to balance data requests fairly among the views. The scheme assumes that if a node receives a fair share of data, this also implies that requests on items are fairly distributed. According to the uniform environment nodes have equal weights and thus should receive the same amount of data items.

In the following we always denote with $h : V \rightarrow M$ the hash function for nodes and with $g : D \rightarrow M$ the function for items. They assumed that the hash functions behave like independently distributed random variables. First all nodes are mapped via $h(v)$ into M and afterwards data elements via $g(d)$. An item d is assigned to the node $v \in V$ which minimizes the distance $|h(v) - g(d)|$, so which is the closest to the item injection point $g(d)$. An example of such an assignment is pictured in Figure 2.2, where v_2 receives d, because $t = |h(v_1) - g(d)| > |h(v_2) - g(d)| = s$.

The major parts of Consistent Hashing are similar to our new DHHT approach, see Sections 2.2, but for better differentiation we describe the DHT functions too. So, how this scheme handles a dynamical and uniform environment can be seen in the following procedure enumeration. Normally, each node is represented by $k \geq 1$ injection points or so called virtual servers, for simplification in description we temporarily set $k = 1$. For a larger k the operations must be extended to cover the impact for nodes adjacent to the copies.

1. $insertItem_{DHT}(d)$: determine $g(d)$ and store d on $v \in V$ if it minimizes the distance $|h(v) - g(d)|$

2. $insertNode_{DHT}(v)$: determine $h(v)$ and the direct neighbors u_L and u_R. Move all items from u_L within the range $[h(v) - |h(v) - h(u_L)|/2, h(v)[$ and $[h(v), h(v) + |h(v) - h(u_R)|/2[$ from u_R to v.

3. $deleteNode_{DHT}(v)$: determine the direct neighbors u_L and u_R of v. Move items in the range $[h(v) - |h(v) - h(u_L)|/2, h(v)[$ to u_L and in the range $[h(v), h(v) + |h(v) - h(u_R)|/2[$ to u_R.

Note, $c(v)$ is limited, a condition for $deleteNode_{DHT}(v)$ and $insertNode_{DHT}(v)$ is that no operation violates the capacity limit of any involved node. An important observation is, that the look up feature is embedded in the DHT scheme. Responsibility ranges of nodes for items are computable and if changes in the environment occur they are fixed locally in the neighborhood. This locality ensures that in case of $insertNode_{DHT}(v)$ items are only moving to the joining or from the leaving node in case of $deleteNode_{DHT}(v)$.

Beside the distribution quality of appropriate hash functions, the approach of [21] suffers under the coupon collector problem, see Section 1.3. Further, the sampling of intervals for nodes and their lengths will deviate from the expectation and there will be nodes receiving intervals that are up to a factor of $\mathcal{O}(\log n)$ times higher than the average size. This is based upon an observation of the maximal distance two nodes will have with high probability if they are placed independently uniformly at random within an unit ring of size 1.

Lemma 1. *On a hash range M interpreted as ring the distance between two node injections is at most $\mathcal{O}(\frac{\log n}{n})$, with high probability.*

Proof. Sketch: Let assume that till now no inserted node is injected in a sub range I of size $\frac{c}{n} \cdot \log n$. Thus, there must exist a pair of nodes with such distance. From this it follows that the probability for a newly inserted node to hit I is $\frac{c}{n} \cdot \log n$ and for hitting the rest of M is $1 - \frac{c}{n} \cdot \log n$. From this it follows straight forward

$$\begin{aligned} P[\text{no } v \in V \text{ hits } I] &= \left(1 - \frac{c}{n} \cdot \log n\right)^n \\ &\leq e^{-\frac{c}{n \cdot \log n} \cdot n} \\ &\leq \frac{1}{n^c} \end{aligned}$$

From this it follows that I will be hit with high probability, and thus a distance between two nodes in M will be at most $\mathcal{O}(\frac{\log n}{n})$ with high probability. □

The resulting problem can almost be resolved by using for every node $k = \mathcal{O}(\log n)$ further independent and randomly chosen injection points in M. Each of these points behave individually as virtual servers with their own responsibility range. The reduction of the deviation from their expectation can be shown by applying Chernoff bounds from Section 1. One can show that the imbalance or error reduces to a small factor of $1 \pm \epsilon$ for any $\epsilon > 0$ [21] [39]. Besides the virtual server concept other approaches have been presented, like multiple choice [29] to overcome the problem of imbalance. The abilities of Consistent Hashing can be summarized in the following corollary, and follow immediately from the proofs in the according article [21].

Corollary 2. *In a dynamical uniform environment the DHT scheme is monotone, space balanced w.h.p., adaptive and $1-$competitive in expectation, if $k = \Omega(\log |V|)$ and at least, the mapping is simple. Eventually, it solves the uniform problem of Definition 15*

Finally some remarks , if we reconsider monotonicity strictly, one problem remains. We have quoted out that DHTs achieve space balance with $k = \Omega(\log |V|)$.

So, nodes are represented by k independent virtual points in M and thus each is responsible for at most k possibly not neighbored subranges in M. For example, if V grows, decreases continually or k alternates, it implies that each node receives or loses a virtual representative. In both cases items will be moved between nodes, which are not involved in the join or leave operation. Eventually, this circumstance violates the monotonicity condition. To over come this behaviour we can choose one of the following assumptions.

Assumption 1. *If (D, V, f) is dynamical, the amount of join and leave operations is nearly equal. Thus, they cause no alternation on the number of virtual nodes.*

Assumption 2. *If the environment (D, V, f) is dynamical we set $|V| \leq N \in \mathbb{N}$, where N is fixed, but sufficiently large. Then we substitute $|V|$ with N and set $k = \mathcal{O}(\log N)$.*

In the following we always make the Assumption 2 and accept a possibly higher granularity and fragmentation of M than needed to achieve the balancing qualities and take the additional overhead for managing the nodes.

Cut and Paste

The *Cut and Paste Strategy* introduced in [11] is similar to Consistent Hashing, and built for an item mapping in a dynamical environment consisting of uniform nodes. Unlike DHT, it uses the hashing scheme only to distribute items among the nodes and not to determine the responsibility ranges for nodes. The goal of substituting the random by a deterministic segmentation is to improve the distribution qualities of DHTs by producing a deviation free share for each node, and coevally preserve the adaptivity in case of dynamics. The node mapping

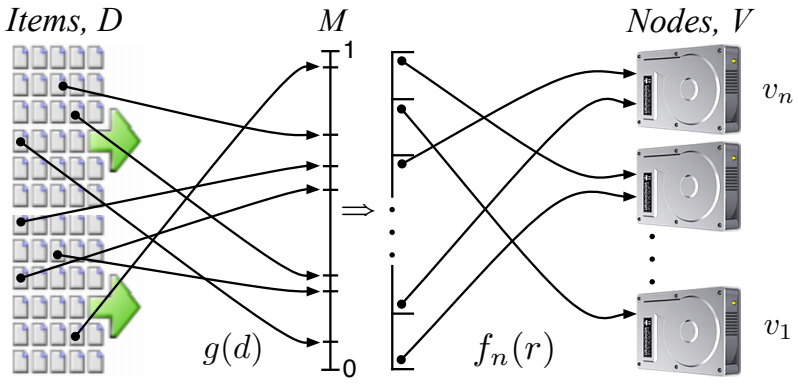


Figure 2.3: The Hashing and Assimilation Phase of Cut-and-Paste

and the segmentation of M is determined with an assimilation function f_n and basically works in the following two phases:

1. For each $d \in D$ use a pseudo-random hash function $g : D \rightarrow M$ to determine an even distribution of item injection points in M.

2. The ssimilation function f_n cuts the hash range M into same sized subranges $r = [lb, ub[$ and assigns each subrange to a node $v \in V$.

Details about f_n can be found in the according articles [11] [39]. An advantage in contrast to DHT is the deviation free transformation from the continuos range to discrete once. The price of the optimal scattering is the order dependancy of participating nodes, because a new node receives a subrange from any present nodes of size $1/n(n+1)$. This order dependancy leads to additional item movements if nodes are removed, because the assimilation function cannot by applied directly. Further, the assimilation function produces for n nodes a fragmentation of $n \cdot (n-1)/2 = \mathcal{O}(n^2)$ and the later a node joins the higher its fused ranges are scattered over M. Finally, the abilities of the Cut and Paste Method can be summarized in the following corollary.

Corollary 3. *In uniform and dynamical environments the Cut and Paste Strategy is space balanced, adaptive and 1-competitive in expectation and monotone for joining nodes. It is 2-competitive in expectation for leaving nodes. It does not solve the Uniform Problem completely, because it violates Definition 15.3 for leaving nodes .*

The most difference of both presented approaches is the transformation from the continuous hash range into discrete subranges. Of course, the deterministically segmentation of Cut and Paste has advantages, like avoiding the deviation produced by random sampling, which is an issue if nodes have restricted recourses. Nevertheless, there are results proving that this distance deviation can be kept constant, with high probability and with slightly changes in the original DHT scheme [33]. To show this, the authors of have considered the ration of the largest and the smallest responsibility range. They denote this ratio as *smooth*, if it is constant. In a distributed environment, where DHTs are very popular the preconditioned knowledge about the order of join and leave operations results into communication and the order dependancy requires synchronized node operations. For large networks such kind of global communication behavior might be a disadvantage. This can be avoided by DHT, but in a SAN such global knowledge is given. Finally the worse competitive ration for leaving nodes and that this operation is not monotone, favors the DHT method. The unwanted sampling deviation of DHTs can be kept small with multiple choice, which is recommended for both methods to suppress random side effects like the Coupon Collector Effect.

2.1.4 The Heterogeneous Case

The main insufficiency of the presented and other uniform distribution strategies is the artificial restriction to node uniformity. Commonly, nodes are not uniform, especially in storage environments the capacity of devices is ongoing growing and an allocation strategy has to respect this technological progress. Currently most allocation strategies handle heterogeneity by dividing huger devices manually into smaller same sized partitions. If the partition size $c(partition)$ is equal for all devices, then the heterogeneity in capacity is resolved. The best can happen is that $c(partition)$ is a divider of the device capacity $c(v)$. In the worst case the remaining capacity of v has nearly the size of $c(partition)$ and is left unused. Especially substituting a device by a newer one is not that easy, if the failing one is old and the substitute should be equal. Such restrictions are inherited if the allocation scheme uses RAID directly to utilize devices, see 1.1.1. In this section we consider heterogeneous node sets and strategies trying to handle heterogeneity more flexible and efficiently than traditional approaches.

According to Section 2.1.2 we adapt Definition 15 to respect environments (V, D, f_W) consisting of heterogeneous nodes. In our case, we restrict heterogeneity to a single attribute, so nodes differ only in storage capacity, thus $|W| = 1$. For remembrance, the weight of $v \in V$ is denoted with w_v and its capacity with $c(v)$. Analog to the uniform case the main objective is the fair and balanced item al-

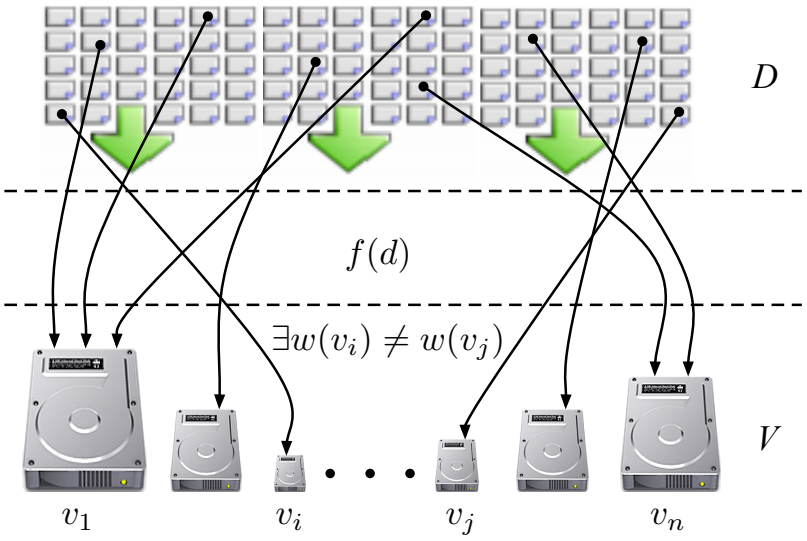


Figure 2.4: The Heterogeneous Case

location on limited resources, where fairness means the relative usage of each node at any time should be the same. Additionally, we assume that items are having same access frequencies, thus such a allocation spreads transient requests weighted among V. Beside capacity allocation this is good if the bandwidth of nodes is not the bottleneck or increases analog with their capacity. An example of such an environment consisting of different sized nodes is pictured in Figure 2.4. However, all aspects discussed at the beginning of Section 2.1 regarding heterogeneity are still not covered completely in this scenario, which leads us to the following problem definition:

Definition 16. *Let (V, D, f_W) be a dynamical environment, where V consists of heterogeneous nodes and D of uniform items. A mapping function f_W solves the heterogeneous problem, if the following terms are fulfilled.*

1. *The mapping of $d \in D$ via f_W is simple. This is the case if f_W uses as input V, W and d without the knowledge of $\bigcup_{v \in V} f_W^{-1}(v)$.*
2. *The mapping is space balanced or fair if for any $v, u \in V$ $\frac{w_v}{f_W^{-1}(v)} \approx \frac{w_u}{f_W^{-1}(u)}$, so each node receives relative to its size the same fraction of D*
3. *The mapping function f_W is adaptive in case of changes in $|V|$ and monotone for join and leave operations*

Similar to Definition 16 the requirements of Definition 16.1 for f_W implie that the item mapping is independent, means we have to avoid global knowledge and ignore the current state of (V, D, f_W) or future mappings of D. Finally, Definition 16.3 implies that all these properties should be preserved after changes in the environment and especially in V have occurred.

2.1.5 The Heterogeneous Case and Solutions

In this section we regard existing strategies, able to solve the heterogeneous distribution problem from Definition 16 partially or completely. Further, we will introduce our solution for the heterogeneous case. Hence we consider our method as a generalization of the original DHT approach, we will also show the equality of our model if the environment consists of uniform nodes. There are several solutions we do not consider and which are coping the demands of heterogeneous capacity allocation in a storage area or cluster networks, [54] [27]. However, strategies we present have in common with ours that they make use of DHTs or likewise hashing schemes. A reason for using DHTs is the aim to overcome problems induced by dynamics, and to do this efficiently, especially in combination with heterogeneity. Unlike most strategies they can handle joining or leaving nodes with less restrictions or costs on these operations.

Reaching Consistency

From Section 2.1.3 we know that DHTs are capable of handling dynamical behavior of nodes. Handling heterogeneity with DHTs and keeping the positive behavior in case of dynamics would be convenient in many areas. Therefore, we have to overcome the major restriction of DHT, nodes are considered to be equal, in our case equipped with the same storage capacity. Anyhow, one can provide with each uniform strategy a heterogeneous solution based on normalization. The idea of such a *Naive Solution* is to resolve heterogeneity with multiple uniform nodes and is described in the following theorem.

Theorem 2. *Any mapping scheme which solves the Uniform Distribution Problem of Definition 15 can be used to solve the space balance condition of Definition 16.2.*

Proof. We know based upon the uniform solution each node $v \in V$ will receive $\lfloor |D|/|V| \rfloor$ items. So we modify V to V' and redefine the number of nodes in such way the each node can pick a number of virtual nodes in V' proportional to its capacity. Than we are done. Therefore we define a normalization factor for every node and determine the needed number of virtual nodes. For simplification and without loss of generality we assume that $\forall v \in V, c_v/c_{min} \in \mathbb{N}$ is true. Let $c_{min} = min(c(u)|u \in V)$ denote the node with the least capacity in the environment, than a normalization factor is defined by $p = 1/c_{min}$. This leads for each node to a virtual node counter of $1 \leq p_v = c(v) \cdot p \in \mathbb{N}$. Than the sum of all normalization counters is $|V'| = \alpha = \sum_{v \in V} p_v$. As long as we treat each virtual node individually, the uniform strategy guarantees that each $v \in V'$ will receive at most $\frac{|D|}{\alpha}$ items. Thus, a node owning p_v virtual nodes will receive at most

$$p_v \cdot \frac{|D|}{\alpha} = p_v \cdot c_{min} = \frac{c(v)}{c_{min}} \cdot c_{min} = c(v)$$

□

DHTs already make use of the virtual node concept to suppress side effects derived by random sampling [21], see Section 1.3. This concept can simply be extended by techniques from Theorem 2 to cope heterogeneity by generating a fitted environment. Therefore, we define V' consisting of $|V'| = \mathcal{O}(\log(|V|) \cdot \alpha)$ virtual nodes. This is needed, because each node injection is treated independently and each node in DHTs needs at least $k = \mathcal{O}(\log(n))$ virtual nodes to achieve its expectation. So, based upon the heterogeneous distribution objective it remains that each node increases its amount of virtual nodes by p_v and k.

Nevertheless, there exists a tradeoff in the determination of V', especially in the heterogeneous case. The smallest nodes in a naive solution using DHT will

become overloaded if k is to small. On the other hand a larger number of virtual servers implies a higher overhead, but less deviation. So if $p_v = c_v \cdot p \cdot k$ is used, we can expect good results with $k \geq \log(n)$. Additionally, because c_{min} is allowed to change, the overhead is dominated at least by c_{min}. All this can lead to an inefficient relation of $|V|$ and $|V'|$, but optimistically only an asymptotical overhead of $\mathcal{O}(\log n)$ remains to organize V. Then the applicability of normalization depends on the expected weight distribution in V.

Such a simple scheme is used in CFS [14] to balance the load among heterogeneous server capacities. A further load balancing scheme called Y_0, based upon DHTs and virtual servers to resolve heterogeneity [17]. They have show that their Y_0 protocol is able to reduce the imbalance of Chord [48] from $\mathcal{O}(\log n)$ to less than 4, but without increasing the number of links a node needs to maintain. For more details on general Ball into Bins like models with heterogeneous bins we refer to another articles [53] [49]. The authors have investigated the impact of multiple choice, if bins are not sampled by an uniform distribution. They have shown tight upper and lower bounds on the number of d choices needed to achieve a balanced allocation. So with an adequate d and if $|D|$ is arbitrarily lager than $|V|$, the maximum load of a bin will be $|D|/|V| + \mathcal{O}(\log\log|V|) + \mathcal{O}(1)$, with high probability. Eventually, if multiple choice is included the additional overhead of the heaviest bin becomes independent on the number of thrown balls.

If we regard the normalization under the conditions of Definition 16 we can conclude that using DHT provides a solution which inherits all attributes from the uniform case, with one exception. Even under the Assuption 2 we can not guarantee the monotonicity of this mapping. While it is an issue to keep the managing overhead for nodes as small as possible, we have to choose the normalization factor as small as possible too. Since we cannot predict c_{min}, the total number of virtual nodes a node uses then depends on it. Finally if c_{min} changes, the problem we have tried to overcome with Assumption 2 occurs again. The random injection of nodes in M leads to specific adjacencies of nodes. If the number of virtual nodes for each node changes, current adjacencies will be destroyed and items are moved in between newly created adjacencies. Finally, the number of items moving can be large, even if the absolute change in C_V is small. All this initiated by the change of the node defining c_{min}!

Corollary 4. *In a dynamical environment (V, D, f_{norm}) consisting of heterogeneous nodes with $|W| = 1$, the Naive Solution based on DHTs is space balanced in expectation and adaptive, 1-competitive and monotone only if $k \cdot \alpha$ is not allowed to change.*

In the next section we present strategies trying to overcome the overhead dependancy on the capacity distribution. The first method decrease the overhead to

a reasonable and assessable size, but for managing the environment and resolving heterogeneity it still needs an uniform strategy.

Share

The Share strategy is one of the first methods which was capable of handling a dynamic storage environment consisting of a node set heterogeneous in a single attribute [12]. Share uses a DHT scheme to resolve the heterogeneous problem by reducing it to a uniform problem. In contrast to the naive normalization the uniform subproblems do not depend on the smallest node. So the remaining overhead to bound heterogeneity stays in a manageable amount. Similar to DHT, Share uses a double sided hashing scheme to transform the heterogeneous problem to a uniform one. Nodes are represented via ranges of a specific length. The length of a node depends on its relative capacity and a stretch factor. Both together should ensure the coverage of the hash range M, which is here interpreted as a unit ring. The starting points of the node ranges are mapped via a hash function into M. Each starting and end point of a node defines a frame of a certain length in M with the other ranges overlapping. Within each frame the nodes are considered to be equal. The fundamental idea is that larger nodes are participating in more frames than smaller once, which leads to the following two staged item mapping. First an item is mapped with into M, which determines the ap-

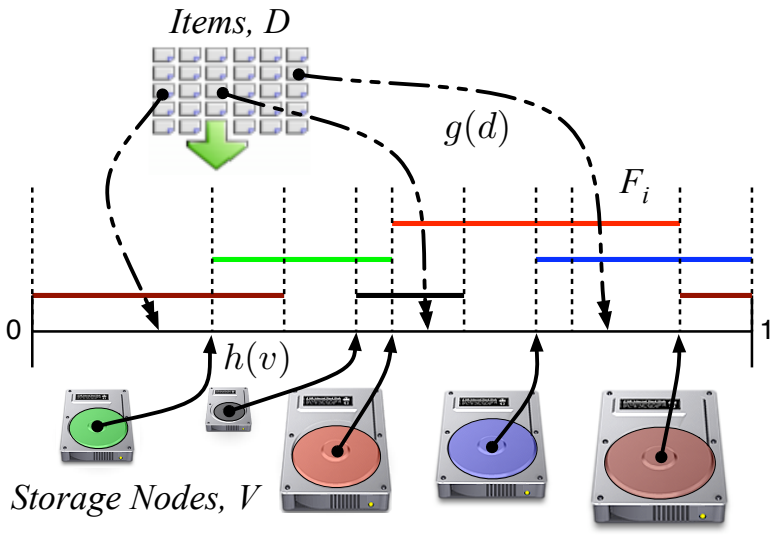


Figure 2.5: Share, reduction to the uniform case

propriate frame and its nodes for the further assignment. Within such a frame the item is distributed with an uniform scheme like DHT to identify the final destination of an item. An example of this mapping scheme is pictured in Figure 2.5 and the abilities of Share are summarized in the following corollary. The analyses are made under the assumption that the maximal number of nodes ever existing in such a network, denoted with N is known. Further, since Share uses a monotone strategy to resolve the uniform problem e.g. Consistent Hashing and because of the lazy update strategy Share is monotone. The lazy update strategy of Share adapts the heterogeneous capacity distribution only if the changes exceed a certain constant.

Corollary 5. *Within a dynamical environment consisting of heterogeneous nodes where* $|W| = 1$*, the Share strategy for a given stretch factor* $s = \mathcal{O}(\ln N)$ *is space balanced, monotone and adaptive with a* $(2+\epsilon)$*-competitiveness for any* $\epsilon > 0$ *if a node joins or leaves the systems.*

Finally, a note on Share [12]. Since the runtime of $\mathcal{O}(1)$ is wanted to map an item and DHTs are used the following space complexity results:

$$\mathcal{O}(2h + s \cdot (n + 1/\delta) \cdot Space_{Uniform}) = \mathcal{O}(n^2) \tag{2.9}$$

This can be followed by the article and argued by the space consumption of DHTs and the number of frames crated to carry out the reduction to the uniform case.

The following strategy copes the same heterogeneous distribution problem, in contrast to Share it resolves heterogeneity directly, without relying on an uniform solution.

Sieve

Sieve is a further method capable of handling a dynamic storage environment, consisting of heterogeneous nodes with $|W| = 1$ [12]. The main enhancement of Sieve is to resolve heterogeneity directly, but to accomplish this it uses more steps for an item assignment than Share. Similar to the Cut and Paste approach, see Section 2.1.3, it uses hash functions only to map items into a hash range M, but not for nodes. To assign nodes to responsibility ranges M is subdivided into consecutive disjoint subranges I_i of size $1/n' = 1/2^{\lceil \log n \rceil + 1}$. Than the nodes are mapped uniquely and distinctively to the subranges, such that halve the subranges are free. Thus, the number of subrange a node occupies is

$$v_{counter} = \left\lceil \frac{w_v/2}{2^{\lceil \log n \rceil + 1} \cdot \sum_{u \in V} w_u} \right\rceil = \left\lceil \frac{w_v}{2^{\lceil \log n \rceil + 2} \cdot \sum_{u \in V} w_u} \right\rceil \tag{2.10}$$

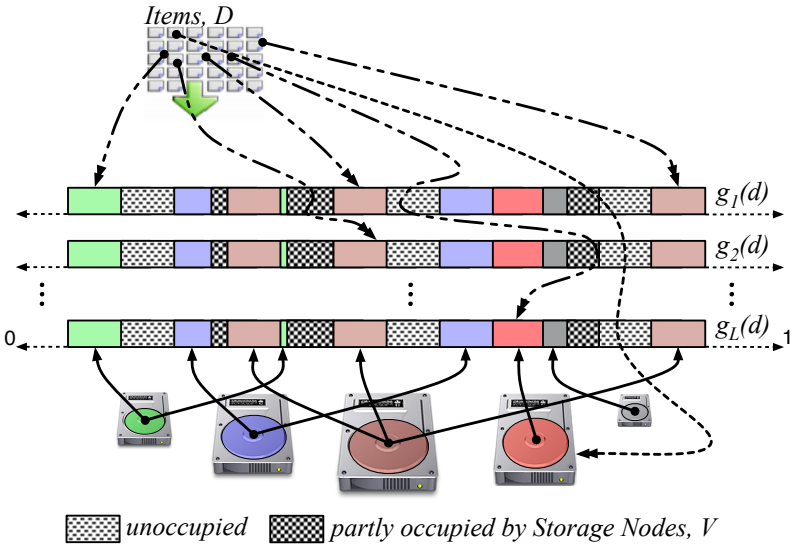


Figure 2.6: Sieve, multiple hash functions

and if $v_{counter} \cdot 1/n' > w_v / \sum_{u \in V} w_u$, v occupies one interval only partially, beginning from the start point of I_i. This partial assignment is necessary to keep the comparative node size. An interesting aspect of Sieve is, that the order of the mapping is completely irrelevant here. The continual item to node mapping is done by L $L-way$ independent has functions $g_i : D \rightarrow M$ as follows:

1. $insertItem_{Sieve}$: Set $i = 1$ and while $g_i(d)$ hits an unoccupied I_j or an unoccupied area in I_j and $i \leq L$ increment i. If $i \leq L$ store d on the node owning I_j, which $g_i(d)$ has hit, else store d on the $fallbacknode$.

Eventually, if the mapping is not successfully after L steps the node is assigned to a previously defined fallback or safety node v_{fb}. Initially this is typically the largest node in V, and may change during runtime. Important here is that the fallback node uses an adapted weight, according to the amount of items probably falling through the sieve. A node to subrange mapping example and the item mapping using L hash functions, including a fall back bin, is pictured in Figure 2.6.

Corollary 6. *Within a dynamical environment consisting of heterogeneous nodes where $|W| = 1$, the Sieve strategy is space balanced and adaptive with a $(2+\epsilon)$-competitiveness for any $\epsilon > 0$ if a node joins or leaves the systems, by using $L = \mathcal{O}(\log n)$ $L-way$ independent hash functions.*

Finally some notes on Sieve. To manage the nodes and the item the strategy needs $\mathcal{O}$(n) plus L $L-way$ independent hash functions. The fragmentation of the hash range M is $2^{\lceil \log n \rceil + 1}$. From the algorithm we can deduce a worst case runtime of $L+1$ for an item assignment. The major advantage of Sieve is the independence of an uniform strategy and if we slacken the requirements on the hash functions, it is also easy to implement.

2.2 Distributed Heterogeneous Hash Tables

In this section we present our double sided weighted distributed hashing approach aka. DHHT [43]. It mainly consists of two different methods and it can be considered as a generalization of the original Consistent Hashing approach [21], we have presented in Section 2.1.3. As seen, the original approach handles heterogeneity only if some kind of normalization is used, see Section 2.1.5. The motivation of this method was to extent the original uniform approach in such way, that on the one hand the two sided hashing scheme is completely persevered and on the other hand the scheme keeps the heterogeneous representations of nodes independent from each other. So, if for any node $c(v)$ or if V changes, the representation of the unchanged part in the environment should be kept untouched. Such an attribute would be eligible for a distributed strategy, because responsibility decisions for items will not depend on the knowledge about the consistency of V, but on the negotiating nodes only. We will see later what this means.

Remember, satisfying these requirements for resolving heterogeneity is not self-evident. Other approaches like Sieve are using a single sided hashing and up to $L+1$ additional steps to assign an item. Share needs an additional structure to resolve heterogeneity, which sometimes needs to be adjusted.Global dependencies on the node representation are suppressed until these effects are not insignificant any more and will have an impact on the space balance. In a distributed environment fixes in the node representation mostly lead to global communication and item reassignments. Furthermore, they will make an asynchronous item assignment difficult if a node does not know the current representation of the environment.

2.2.1 An Alternative Representation of Consistent Hasing

In the following we transform the original DHT model into an alternative, but equal representation, we denote as DHT*. The aim is to obtain an independent representation of nodes extendable for a heterogeneous solution and to keep the positive attributes of DHT. From now on we denote node injection points with

$r_v \in M$ and item injections with $r_d \in M$. Further, we still assume that any injection points are obtained by using an adequate hash function in the hash range M, which is interpreted as a ring. This implies that the length of a subrange $I_v \subseteq M$ owned by v is equal to its expectation to be hit by any further injection point. As a reminder, the item assignment in DHT is done by choosing the closest node injection point r_v to r_d, and DHT resolves directly the uniform case only, see 2.1.3. We can assume that $c(v) = c(u)\ \forall v, u \in V$ and we say there exists a globally known constant c.

Our transformation requires the assignment of two linear functions $f_{L,v} : M \to \mathbb{R}$ and $f_{R,v} : M \to \mathbb{R}$ to each nodes $v \in V$. Additionally we claim that for all nodes the gradient value of their functions is only distinct in sign, where the index L marks the negative gradient and $f_{L,v} = f_{R,v}|y = 0$ marks r_v. Apparently, the input of both functions are item injections $r_d \in M$. Further, because M is a unit ring, if $f_{L,v}$ leaves the range at 0, it comes back at 1 with same gradient and the height it left at 0. This is analogue for $f_{R,v}$. So D_{DHT^*} consists of four linear functions, two starting from r_d with gradient $1/c$ and $-1/c$, one starting at $r_v - 1$ with $1/c$, and one starting from $1 + r_v$ with $-1/c$. This results in the following distance function $D_{DHT^*} : [0,1[\times[0,1[\to \mathbb{R}$ for a node v to determine the height of an item d:

$$D_{DHT^*}(r_v, r_d) = \begin{cases} (r_d - r_v)/c & \text{if } r_d \geq r_v \wedge |r_d - r_v| \leq 1/2 \\ (r_d - (r_v - 1)/c & \text{if } r_d \leq r_v \wedge |r_d - r_v| > 1/2 \\ -(r_d - r_v)/c & \text{if } r_d < r_v \wedge |r_d - r_v| \leq 1/2 \\ -(r_d - (r_v + 1))/c & \text{if } r_d > r_v \wedge |r_d - r_v| > 1/2 \end{cases} \tag{2.11}$$

On can see that $D_{DHT^*}(r_v, r_d)$ always returns the minimal positive value at position $r_d \in M$ for a node owning r_v and a fixed c. This leads to the following function to determine the minimal height over all nodes and thus all functions in M, which coevally identifies the node which is responsible for d.

$$H_{DHT^*}(d, V) := \min(\{D_{DHT^*}(r_v, r_d) \mid v \in V\}) \tag{2.12}$$

This distance functions allows us to define the following DHT* algorithms to determine a node insert or delete and an item mapping in M. We do not consider the deletion of items.

1. $insertItem_{DHT^*}(d)$: calculate r_d and store d on the $v \in V$, where $D_{DHT^*}(r_v, r_d) = H_{DHT^*}(d, V)$. Note, therefor we only have to find the left and right neighbored nodes v_L and v_R of r_d. In case of $D_{DHT^*}(r_{v_L}, r_d) = D_{DHT^*}(r_{v_R}, r_d) = H_{DHT^*}(d, V)$, choose v_L.

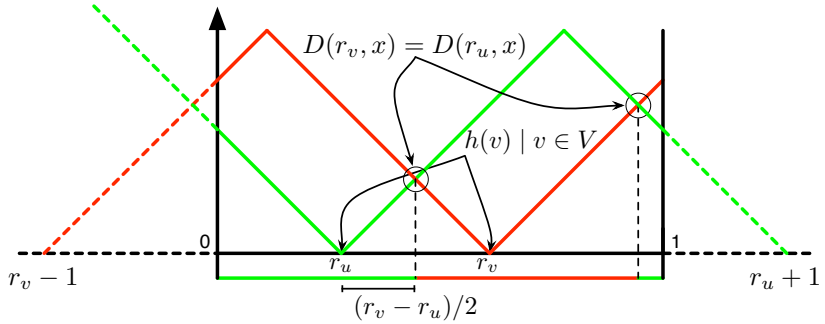


Figure 2.7: DHT*, an Alternative Reprensentation of DHT

2. $insertNode_{DHT^*}(v)$: calculate r_v and determine the direct neighbors u_L and u_R of v. Move all items $d \in f^{-1}(u) \mid u \in \{u_L, u_R\}$ to v, where $D_{DHT^*}(r_v, g(d)) < D_{DHT^*}(r_u, g(d))$ is true.

3. $deleteNode_{DHT^*}(v)$: Move all $d \in f^{-1}(v)$ to the direct neighbor u_L where $D_{DHT^*}(u_L, g(d)) < D_{DHT^*}(u_R, g(d))$ is true and and if false to u_R.

Without loss of generality, but for simplification we have not considered node copies. This is acceptable, because node copies must be treated individually and independently in the DHT scheme.

Different to DHT, DHT* uses values defined by $D_{DHT^*}(r_v, r_d)$, which can be considered as the height an item d receives on a node v. This height depends on the gradient $1/c$ (or in the uniform case as substitute for $c(v)$, w_v) and r_v. That minimizing the height is the same as minimizing the distance in the uniform case, shows the following lemma:

Lemma 2. *DHT is equal to DHT*, since c is known $\forall v, u \in V$ is true.*

Proof. This follows directly by simple geometry facts. Let f_1 and f_1 be two linear functions, one with $1/c$ and the other with $-1/c$ as gradient.

1. f_1 and f_2 are axially symmetric according to an axis aligned parallel to the y-axis through their intersection point $P(f_1, f_2)$.

2. If $|P(f_1, f_2).y| > 0$, than $P(f_1, f_2).x$ halves the distance between $x_1, x_2 \in M$, where $f_1(x_1) = f_2(x_2)$, because $\frac{x_1 - x}{c} = -\frac{x_2 - x}{c} \Longrightarrow \frac{x_1 + x_2}{2} = x$.

3. linear functions with the same gradient have no intersection, if they are not identically.

We know if f_1 and f_2 are owned by to different nodes $u, v \in V$, an item d is assigned to the closest node. This is, because the height of the closer node is less than the height of the farer one. From 3. we know that the intersection with the minimal height is caused by an adjacent node. Clearly, the domination alters at $P(f_1, f_2).x$, which coevally halves their distance. In case of $r_d = P(f_1, f_2).x$, then $D(r_v, r_d) = D(r_u, r_d)$. This mapping decision can be made via definition, which shows the equality of DHT and DHT* in the uniform case. Hence M is used as a unit ring this holds for any intersection of f_1 and f_2 □

The geometric construction of the distance function $D(r_v, r_d)$, interpreted in a ring, which builds up $H_{DHT^*}(d, V)$ and the conclusion of Lemma 2 can be reconstructed from Figure 2.7. The example uses only two nodes, but the effect of halving the distance between adjacent nodes is independent in the number of nodes. In contrast to halving, the perfect segmentation of M at any position is only achievable up to two nodes.

Finally, some important notes on $D_{DHT^*}(r_v, r_d)$ and $H_{DHT^*}(d, V)$ if the linear functions f_1 and f_2 should be substituted by any other functions. The node functions must have the following attributes. First, they must be continuous and strictly monotonic, one increasing and one decreasing over M. This is needed to get intersection points and avoid intersection ranges and thus obtain explicit responsibility ranges. Second, they must be axially symmetric according to an axis aligned parallel to the y-axis through r_v. This is needed, because for any pair wise distinct $v, u \in V$ with same weights f_{v_L} is parallel to f_{u_L} and analogues for f_{v_R} and f_{u_R}. From this it follows that f_{v_L} and f_{u_R} are also axially symmetric according to the axis running parallel to the y-axis and through $P(f_{v_L}, f_{u_R})$ and $P(f_{v_L}, f_{u_R}).x$ halves the distance between r_v and r_u, which was needed to show the equality of both models.

In the the following sections we introduce our DHHT approach, including two methods, both capable of resolving one heterogeneous attribute of nodes in a dynamic environment. Both variants use a double sided hashing approach and the item to node mapping specified in DHT*. As we know from Section 2.1.5, normalization is a disappointing opportunity to resolve heterogeneity, as long as a managing overhead for nodes and migration of items caused by changes is an issue. If we replace DHT with DHT*, these costs are still existing. However, DHT* has the potential to overcome the overhead costs depending on the capacity distribution and the resulting non monotone behavior.

2.2.2 The Linear Method

In this section we present the first DHHT variant, the *Linear Method*, in the following abbreviated with $DHHT_{Lin}$. It is designed for resource allocations in dynamic environments $(V, D, f_{DHHT_{Lin}})$ and resolves heterogeneity of nodes according to a single attribute, so $|W| = 1$. Our aim is to overcome the node dissimilarity directly, without using an uniform strategy, and to avoid global adaptions in the node representations if the environment changes partial, and to impede item movements otherwise. This prevention is reasonable, because under practical considerations significant capacity changes and the domination by the smallest are typical cases beside node failures or small capacity expansions. For instance, a storage environment grows steadily by inserting larger devices or older and smaller once are removed, especially if they fail and typically replaced by larger once. So changes caused by exchanging the smallest node as well as capacity expansion will occur. In the following we show that we can resolve heterogeneity by modifying the distance function of DHT*.

If we examine D_{DHT^*} and $H_{DHT^*}(d, V)$, one can see that the intersection of nodes $P(f_{u_R}, f_{v_L}).x$ is shifted away from the halve distance if individual and unequal node gradients are used. These shifts resize the responsibility ranges and thus the probability to receive an item changes too. In Figure 2.8 we exemplify this effect with two nodes, if the gradient c in D_{DHT^*} is substituted by $w_v := c(v)$, $\forall v \in V$, by using a weight relation $\alpha := w_v/w_u = 2$. In the example the node injection generates in M the range $[r_u, r_v]$, which is likewise subdivided into $I_u^1 := [r_u, \frac{\alpha \cdot r_u + r_v}{1+\alpha}]$ belonging to u and $I_v^1 := [\frac{\alpha \cdot r_u + r_v}{1+\alpha}, r_v]$ belonging to v, by their intersection at $P(f_{u_R}, f_{v_L}).x = \frac{\alpha \cdot r_u + r_v}{1+\alpha}$. This results into the wanted ratio of

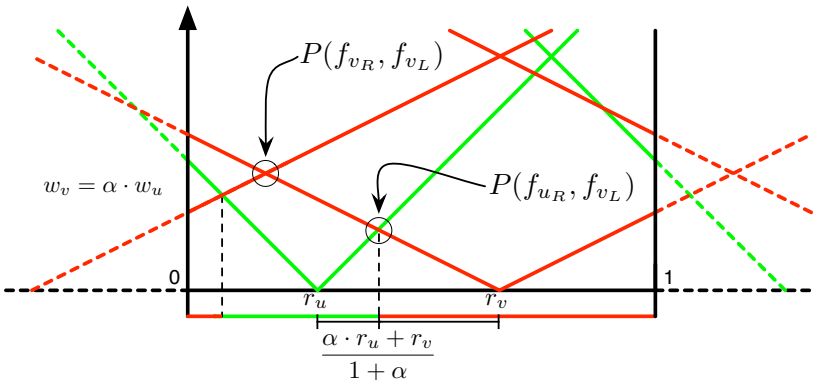


Figure 2.8: Perfect Segmentation with $|V| = 2$ and $\alpha = 2$

$I_v^1 / I_u^1 = \alpha$. This ratio is the same for the segmentation of $[0, r_u] \cup [r_v, 1]$ resulting in $I_u^2 := [P(f_{u_L}, f_{v_R}).x, r_u]$ and $I_v^2 := [r_v, 1] \cup [0, P(f_{u_L}, f_{v_R}).x]$. Together we have $I_v := I_v^1 \cup I_v^2$ and $I_u := I_u^1 \cup I_u^2$ with $|I_v|/|I_u| = \alpha$. Thus, the probability for a node to receive an item randomly injected in M is

$$\frac{|I_v^1| + |I_v^2|}{|I_v^1| + |I_v^2| + |I_u^1| + |I_u^2|} = \frac{|I_v|}{\sum_{i \in \{u,v\}} |I_i|} = |I_v| = \frac{c(v)}{\sum_{i \in V} c(i)} \tag{2.13}$$

The interesting part of this substitution is that c depends directly on the attribute we try to resolve and not on a relative system value, thus node weights and their representation do not dependent on each other.

Note Figure 2.8 only exemplifies how segments in M are constructed and in contrast to the uniform case, the segmentation by two nodes is not always perfect. If r_v and r_u are determined by random the probability to achieve a segmentation with properties of Equation 2.13 decreases, if the ratio α increases. Assuming $w_u \leq w_v$, than the scope where a placement of r_u creates a perfect segmentation is ranged by the following condition

$$\frac{|P(f_{v_R}, f_{v_L}).x - r_v|}{|P(f_{v_R}, f_{v_L}).x - r_u|} = \frac{1}{2} \cdot |P(f_{v_R}, f_{v_L}).x - r_u|^{-1} \geq \alpha \tag{2.14}$$

which is fulfilled if the condition $P(f_{v_L}, f_{u_R}).y, P(f_{v_R}, f_{u_L}).y \leq P(f_{v_L}, f_{v_R}).y$ is true. So, since $w_v \geq w_u$, then $P(f_{v_R}, f_{v_L}).y$ decreases with an increasing w_v, which leads to

$$\begin{aligned} \frac{1}{2} \cdot w_v^{-1} &= \frac{\frac{1}{2} - r_u}{w_u} \\ r_u &= \frac{1}{2} - \frac{1}{2 \cdot \alpha} \end{aligned}$$

So the size of the scope surrounding $P(f_{v_R}, f_{v_L}).x$, where r_u must fall into via random choice, is equal to the probability of the event to occur and can be determined in the two node example by

$$P[r_u \text{ segments optimal}] = (\frac{1}{2} - r_u) \cdot 2 = \left(\frac{1}{2} - (\frac{1}{2} - \frac{1}{2\alpha})\right) \cdot 2 = \alpha^{-1}$$

Another crucial problem is the imbalance caused by dynamical changes in V and its heterogeneity distribution. Similar to DHT the system has to handle the insertion or removal of nodes over time. In our two node example a further addition of a node will automatically cause an unwanted imbalance, this is analog, if a previously perfect segmentation is destroyed by removing a node. Hence previously assigned items keep their positions in M some will be assigned to the new

device or the adjacency of a removed node. In contrast to the uniform case the amount of moved items depends on the weight ratio of adjacent nodes and not only on the distribution quality of nodes among M. So, independent of the balance at a specific moment dynamical changes potentially destroy this quality over time. The only way to keep the balance without increasing the model complexity, is to change r_v for nearly each node in such way that the distances reflect the wanted distribution again. This reactive repairing strategy is unwanted, because the amount of item exchanges among nodes can become very high compared to the capacity change in the environment.

The results in following subsections will show how such a scheme can provide a fairly balanced distribution of data elements. Further, we will see that it is also capable of compensating dynamics and keep balance with minimal effort, concerning data movements or system reorganization by changes.

The Basic Method

As before, given a dynamic environment $(V, D, f_{DHHT_{Lin}})$ consisting of a node set $V = \{v_1, \ldots, v_n\}$ and heterogeneous according to one attribute. The node weights are determined by $W_{DHHT} := \{w(v_i) \in \mathbb{R}^+\}$ reflecting the dissimilarity of V. For instance, if we have one attribute, $w_v = w(v_i) = c(v_i)$. Further we have a uniform item set $D = \{d_1, \ldots, d_m\}$ we want to distribute among the nodes. The mapping function $f_{DHHT_{Lin}}$ determines the rules for an item mapping. Therefore we use the hash functions $r_v := h(v)$, $v \in V$ for the node mapping and $r_d := g(d)$, $d \in D$ for items in M. We still assume that the hash functions behave like independent uniform random variables in the unit range. In contrast to the previous section we simplify the distance function for analyzing purposes. Instead of using two linear functions for each node, the distance function only consists of one linear function per node, each with a positive gradient. So we do not need the axial symmetry property of DHT* any more. Since the objective is unchanged the modification of each node its representation should still result into an imbalance among the nodes, reflecting the weight distribution in W. To decide the item mapping among the nodes we first define the scaled distance function $D_{DHHT_{Lin}} : [0,1[\times[0,1[\times\mathbb{R} \to \mathbb{R}$. Note, we still consider M as a unit ring, so with $r_d, r_v \in M$, this leads to

$$D_{DHHT_{Lin}}(r_d, r_v, w_v) := \begin{cases} \frac{r_d - r_v}{w_v} & \text{if } r_d \geq r_v \\ \frac{r_d + 1 - r_v}{w_v} & \text{else} \end{cases}$$

transformed to a closed form

$$D_{DHHT_{Lin}}(r_d, r_v, w_v) = \frac{1 + r_d - r_v - \lfloor 1 + r_d - r_v \rfloor}{w_v},$$

then by applying Definition 8, we have

$$D_{DHHT_{Lin}}(r_d, r_v, w_v) = \frac{(1 + (r_d - r_v)) \bmod 1}{w_v} \tag{2.15}$$

On can see that $D_{DHHT_{Lin}}$ always returns the minimal positive value at position $r_d \in M$ for a node at r_v with weight w_v. Likewise DHT*, the *Linear Method* should assign an item d at r_d to a node v_i which minimizes the height over all nodes. Therefore we define a function which determines the minimal height to any position $r \in M$ over all nodes. The node defining this minimal height is responsible for d at r_d.

$$H_{DHHT_{Lin}}(r, V) := \min\Big(\{D_{DHHT_{Lin}}(r, r_v, w_v) \mid v \in V\}\Big) \tag{2.16}$$

In contrast to the uniform case, distance functions with different gradients can have intersections with each other and can generate a multiple alteration of the

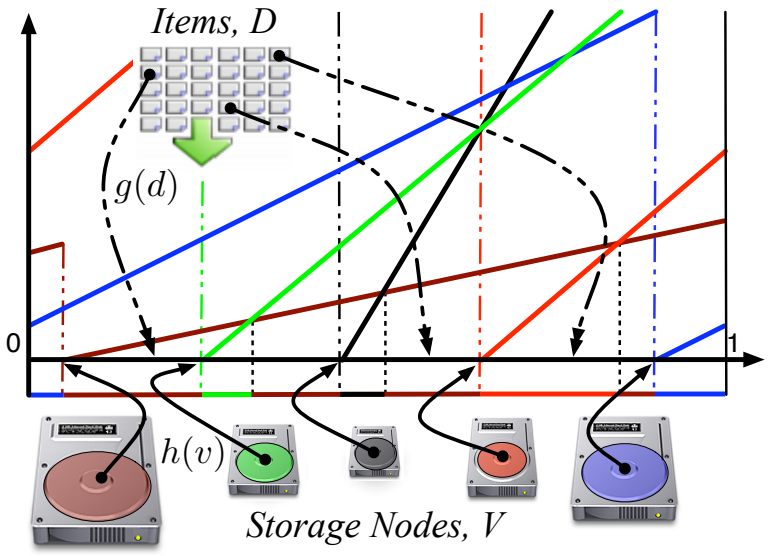


Figure 2.9: DHHT, the Basic Linear Method

minimum according to a single node. Since the minimum of $D_{DHHT_{Lin}}(x, r_v, w_v)$ defines the node which is responsible for the item, each intersection with this minimum is also an alteration of the minimum. This alteration of the minimum can be reconstructed by the domination of the left and brown colored device in Figure 2.9. Finally multiple unique ranges $I_v^i \subseteq M, i \in \mathbb{N}$ can belong to one node v. Further, the distance function of v also defines the maximal height an item receives in this segment. In the following we will denote the intervals defining the minimum over $D_{DHHT_{Lin}}$ the lower envelope of $H_{DHHT_{Lin}}$ and the set consisting all these subranges is denoted by I_{min}. Clearly, the following is true; $\sum_{I \in I_{min}} |I| = 1$ and $\forall I, I' \in I_{min} \wedge I \neq I' : I' \cap I = \emptyset$. Thus, the probability of v to receive an item is determined by $\sum_{I_v^i \in I_v} |I_v^i|$, which is the size of all ranges v is responsible for. These rules for an item assignment leads to the following operations if a node or item is inserted or deleted:

1. $insertItem_{DHHT_{Lin}}(d)$: determine r_d and find the range $I_v^i \in I_{min}$, where $r_s \in I_v^i$. Then store d on the v, because $D_{DHHT_{Lin}}(r_d, r_v, w_v) = H_{DHHT_{Lin}}(r_d, V)$.

2. $insertNode_{DHHT_{Lin}}(v)$: determine r_v and use $D_{DHHT_{Lin}}(x, r_v, w_v)$ to identify intersections $P(u, v)$, $u \in V$, where $P(u, v).x$ in $I_u^i \in I_{min}$, and $P(u, v).y$. Adapt the intersected intervals and remove all completely dominated. Move all items previously assigned by removed intervals or by the changed fraction of intervals to the new node v.

3. $deleteNode_{DHHT_{Lin}}(v)$: $\forall I_v^i \in I_v \cap I_{min}$ determine the lower envelope between $[I_v^i.start, I_v^i.end]$. Insert all new intervals to I_{min} and resize the previously intersected by $D_{DHHT_{Lin}}(x, r_v, w_v)$. Remove $I_v^i \in I_v \cap I_{min}$ from I_{min}. Move all $d \in f^{-1}(v)$ to the corresponding intervals of the changed areas in I_{min}.

As before in DHT or DHT* inserting is nearly the same as deleting an item, because the identification of the appropriate node is major and analog part of the operation. We assume that items are inserted and removed by their keys and the distribution function $g(d)$ is a perfect hash function. Thus deletion of items is evenly distributed over M and thus the probability for nodes to lose an item is the same as for receiving one. Note, all these operations can be implemented in several ways, for instance a central solution can determine in advance the lower envelope and save I_{min} as list sorted among start and end points of intervals. Details on efficient algorithms can be found in Section 2.3.3.

So, we can derive our first property according to Definition 16. One can see that the operation $insertNode_{DHHT_{Lin}}(v)$ has only an impact on those nodes responsible for a removed or changed interval part of the lower envelope. Each

involved node moves items only to the new node, which carries over their ranges and none among others. The same behavior we can observe for $deleteNode_{DHHT_{Lin}}(v)$. A the leaving node only moves items to nodes covering the abandoned ranges of the lower envelope.

Corollary 7. *The Linear Method of DHHT, $DHHT_{Lin}$ is monotone concerning insert or delete operations of nodes.*

However, this monotone behavior does not depend on the heterogeneous node representation within the model. Further, the representation of a node never changes, because it is independent on the number of nodes leaving or joining the environment. The heterogeneity is obtained indirectly. Remember, till here we do not know if this heterogeneity reflects the objected weight distribution.

Let us first take a look at the distribution of the height h according to a position $r \in M$. This, we define in the following lemma

Lemma 3. *Given n nodes and weighting functions $w(i) : \{1, \ldots, n\} \to \{w_1, \ldots, w_n\}$. Then the height $H(r)$ assigned to a position r in M is distributed as follows:*

$$\mathbf{P}[H(r) > h] = \begin{cases} \prod_{i \in [n]} (1 - h \cdot w_i)\,, & \text{if } h \leq \min_i\{\frac{1}{w_i}\} \\ 0\,, & \text{else} \end{cases} \tag{2.17}$$

So the probability for a position and thus an item to receive a specific height depends directly on the minimum defined by the distance function and it is 0 above. In the following theorem we proof the balancing quality of the described mapping scheme.

Theorem 3. *In a dynamic environment $(V, D, f_{DHHT_{Lin}})$ consisting of heterogeneous nodes with $|W| = 1$ and positive weights $w_i := w(v_i)$, the Linear Method assigns an item $d \in D$ to a node $v_i \in V$ with the probability of at most*

$$\begin{aligned} \mathbf{P}[d \text{ assigned to } v_i] &\leq \frac{w_i}{\sum_{u_j \in V \setminus \{v_i\}} w_j} \\ &= \frac{w_i}{(\sum_{j=1}^{|V|} w_j) - w_i} \end{aligned}$$

Proof. Let r_i denote the position, where a data element is inserted. Let H_i denote the height of the data element. Note that all these random variables are independent and uniformly distributed, since we consider only the placement of one data element. Hence, the probability that r_i is at most in the weighted distance h or the element receives at most the height h from node v_i, is described as follows.

$$\mathbf{P}[H_i \leq h] = \begin{cases} 1\,, & h \geq \frac{1}{w_i} \\ h \cdot w_i & \text{else.} \end{cases} \tag{2.18}$$

From this we can determine the probability, that an element receives a height in certain range, like the interval $[h, h+\delta]$ for node v_i and receives greater heights for all other nodes. Therefore, we set $h + \delta \leq \frac{1}{w_i}$ and define the condition $con := H_i \in [h, h+\delta] \wedge \forall j \neq i : \ H_j > h$

$$\mathbf{P}[\text{con is true}] = \begin{cases} 0\,, & \exists j \ : \ h \geq \frac{1}{w_j} \\ \delta w_i \prod_{j \neq i}(1 - h w_j) & \text{else}\,. \end{cases} \tag{2.19}$$

Let $P_{i,h,\delta} := \delta w_i \prod_{j \neq i}(1 - h w_j)$. Now, an upper bound on the probability that an element is assigned to node v_i is given by the sum $\sum_{m=1}^{\infty} P_{i,\delta m,\delta}$. Note we substitute h with $m \cdot \delta$ and obtain

$$\begin{aligned} P_{i,h,\delta} &= \delta w_i \prod_{j \neq i}(1 - m\delta \cdot w_j) \\ &\leq \delta w_i e^{-\delta m \sum_{j \neq i} w_j} \end{aligned}$$

Now let $a := \sum_{j \neq i} w_i > 0$ the sum of weights without v_i. Further, the substitution of h allows us to transform the sum $\sum_{m=1}^{\infty} P_{i,\delta m,\delta}$ into an integral if δ tends to 0. Then we finally obtain

$$\begin{aligned} \lim_{\delta \to 0} \sum_{m=1}^{\infty} P_{i,\delta m,\delta} &\leq \lim_{\delta \to 0} \sum_{m=1}^{\infty} w_i \delta e^{-a\delta m} \\ &= \int_{x=0}^{\infty} w_i e^{-ax} dx \\ &= \frac{w_i}{a} \\ &= \frac{w_i}{\sum_{j \neq i} w_j} \end{aligned}$$

which proves the statement of the theorem □

From this we have a first impression according to the distribution quality of items, but it seems to be disappointing concerning the demanded balancing attribute from Definition 16. The larger a node weight compared to other nodes is, the more it tends to behave like a magnet for an item assignment.

The Use of Copies

In this section we take a look at the use of copies to enhance the balancing property of $DHHT_{Lin}$. Using for each node a number of independent copies is equal to the virtual node technique we already have seen in DHT. There it was used to

ensure that each node receives its expectation, here it can help us to overcome the intrinsic imbalance shown by Theorem 3.

In the following each node will participates in the environment with $\lceil \frac{2}{\epsilon} + 1 \rceil$ copies for some $\epsilon > 0$. We do not formalize this property, but we have to treat each copy separately to ensure independence among the copies. We simply note that the number of nodes needs to be increased by this constant factor, to ensure the following inequality for all $i, j \in \{1, \ldots, n\}$

$$\sum_{i=1}^{n} w_i \leq (1 - \frac{1}{2}\epsilon)^{-1} \sum_{i \neq j} w_i \leq \ (1 + \epsilon) \sum_{i \neq j} w_i$$

This inequality is needed in the theorem below to proof the upper bound of p_i for a node v_i. In the following we abbreviate the sum of all weights with $W_{all} := \sum_{i=1}^{n} w_i$. The following theorem bounds the probability p_i of a node $v_i \in V$

Theorem 4. *In a dynamic environment $(V, D, f_{DHHT_{Lin}})$ consisting of heterogeneous nodes with $|W| = 1$, the Linear Method assigns an item $d \in D$ to a node $v_i \in V$ with probability p_i, if it uses $\lceil \frac{2}{\epsilon} + 1 \rceil$ copies for an $\epsilon > 0$, where*

$$(1 - \sqrt{\epsilon}) \cdot \frac{w_i}{W_{all}} \leq p_i \leq (1 + \epsilon) \cdot \frac{w_i}{W_{all}}$$

Proof. First we proof the upper bound on p_i, which can be shown by using Theorem 3 in combination with the previously defined inequality. This results into

$$\begin{aligned} p_i &= \frac{w_i}{\sum_{j \neq i} w_j} \\ &\leq \frac{1}{1 - \frac{1}{2}\epsilon} \frac{w_i}{\sum_j w_j} \\ &\leq (1 + \epsilon) \frac{w_i}{W_{all}} \end{aligned}$$

Form here, we only have to show the lower bound. Therefore we proof the following lemma which uses the fact that the maximal height in M is restricted by the biggest weight.

Lemma 4. *For all $i, j \in [n]$ with $\epsilon' > 0$ and a height $h \leq \frac{\epsilon'}{\max_i \{w_i\}}$ the following inequality holds*

$$e^{-hW_{all}/(1-\epsilon')} \leq \prod_i (1 - hw_i) \leq \prod_{i \neq j} (1 - hw_i)$$

Proof. To show this we have to remember that for any $k > 1$ the inequality

$$e < (1 + \frac{1}{k})^{k+1} \implies e^{-1} < (1 - \frac{1}{k})^{k-1}$$

is true. So, within the mapping range for any $x \in (0, 1)$ we have

$$1 - x > (1 - x)^{\frac{-x}{1-x}}$$

So we can say for any $h \leq w_i$:

$$(1 - hw_i) > e^{-\frac{hw_i}{1-hw_i}}$$

Note that we have previously set $hw_j \leq \epsilon'$, so we have

$$\begin{aligned} \prod_j (1 - hw_j) &\geq e^{-\sum_{j \neq i} \frac{hw_j}{1-hw_j}} \\ &\geq e^{-\sum_j \frac{hw_j}{1-\epsilon'}} \\ &\geq e^{-hW_{all}/(1-\epsilon')} . \end{aligned}$$

which finally proofs the lemma. □

Again we say that a sufficient condition for an item assignment is $con := (H_i \in [h - \delta, h] \wedge \forall j \neq i : \ H_j > h)$ that a data element is assigned to node v_i is the following:

$$\mathbf{P}[\text{con is true}] = \begin{cases} 0, & \exists j \mid h \geq \frac{1}{w_j} \\ \delta w_i \prod_{j \neq i} (1 - hw_j) & \text{else} \end{cases}$$

let $P'_{i,h,\delta} := \delta w_i \prod_{j \neq i} (1 - hw_j)$ for $h \leq \min\{1/w_j\}$. Now, the sum can be transformed into an integral if δ tends to 0, so

$$\begin{aligned} S &= \lim_{\delta \to 0} \sum_{m=1}^{\frac{\epsilon'}{\delta \max\{w_i\}}} P'_{i,\delta m,\delta} \\ &= \int_{h=0}^{\frac{\epsilon'}{\max\{w_i\}}} w_i \prod_{j \neq i} (1 - hw_j) dh \end{aligned}$$

gives the lower bound on the probability that an element at r_d is assigned to node v_i. So with the height distribution of Lemma 3 and if we set $\epsilon = \epsilon'^2$ we have

$$\begin{aligned}
\mathbf{P}[r_d \text{ assigned to } v_i] &> \int_{h=0}^{\frac{\epsilon'}{\max\{w_i\}}} w_i e^{-hW_{all}/(1-\epsilon')} dh \\
&= (1-\epsilon')\frac{w_i}{W_{all}} \left(1 - e^{-\frac{\epsilon' W_{all}}{(1-\epsilon')\max\{w_i\}}}\right) \\
&\geq (1-\epsilon')\frac{w_i}{W_{all}} \left(1 - e^{-\frac{\epsilon'}{(1-\epsilon')\epsilon}}\right) \\
&\geq (1-\epsilon') \left(1 - \frac{(1-\epsilon')\epsilon}{\epsilon'}\right) \frac{w_i}{W_{all}} \\
&\geq (1-\epsilon') \left(1 - \frac{\epsilon}{\epsilon'}\right) \frac{w_i}{W_{all}} \\
&\geq (1-\sqrt{\epsilon})\frac{w_i}{W_{all}}
\end{aligned}$$

which proofs the lower bound of the theorem. □

So the Linear Method using a sufficient number of copies can provide a balancing, such kind that the probability of a node v_i to receives an item depends proportional on W_{all} and not only on $W_{all} \setminus \{w_i\}$. This closes the gap, where lager nodes tend to receive more items than objected.

The Use of Partitions

Nevertheless, the previous result of Theorem 4 does not imply that every node receives data elements according to the probability, we have determined. Based upon the random sampling of positions in M, $r_v \in M$ once assigned to a node v never changes. So as long as none of the nodes is removed or newly inserted, the resulting intervals assigned to each also remain the same. Finally, because p_v corresponds to the size of intervals assigned to each node it is also fixed after nodes are settled in M. Even in the balanced case, where $\forall v \in V\ |I_v| = \frac{w_v}{W_{all}}$ the Coupon Collector effect still occurs. Eventually, some nodes will receive intervals of a size which are up to a factor of $\mathcal{O}(\log n)$ larger than desired, which directly results into an unwanted imbalance. One technique to resolve this is using more copies, like DHT does, but we suggest and introduce partitions.

Therefore, we partition the hash range into $\mathcal{O}(\log n)$ partial intervals of equal size. For each partition we apply the node mapping of $DHHT_{Lin}$ individually. So the nodes $V = \{v_1, \ldots, v_n\}$ are mapped to each of these continuous set $M_1, \ldots, M_k$

with $M_i = [(i-1)/k, i/k)$ and $k = \mathcal{O}(\log n)$ including node copies. Than all items $d \in D$ are mapped among the whole hash range $[0, 1)$ using hash function for items.

Theorem 5. *For all $\epsilon, \epsilon' > 0$ and $c > 0$ there exists $c' > 0$ such that when we apply the Linear Method to n nodes using $\lceil \frac{2}{\epsilon} + 1 \rceil$ copies and $c' \log n$ partitions, the following holds with high probability. Every node $v_i \in V$ receives all data elements with probability p_i such that*

$$(1 - \sqrt{\epsilon} - \epsilon') \cdot \frac{w_i}{W} \;\leq\; p_i \;\leq\; (1 + \epsilon + \epsilon') \cdot \frac{w_i}{W}$$

Proof. This result follows by Theorem 4 and applying Chernoff bounds from Theorem 1. □

The advantage of applying partitions in contrast of using copies is that nodes of different partitions can not compete against each others. The partitions are similar to independent and multiple instances of the Linear Method using copies. On closer examination the number of partitions should not change frequently, otherwise the monotonicity is endangered. If the partition borders are adjusted, because of newly or collapsing subranges some nodes will need to be re injected, followed by item reassign. This will lead to a non monotone behavior and we will obtain similar effects as we have seen in Share or Sieve if the model adjusts. Since we use $\mathcal{O}(\log n)$ partitions we can keep the monotonicity if we use Assumption 1 or Assumption 2 and set the number of partitions to $\mathcal{O}(\log N)$.

The Limits of the plain Linear Method

In this section we show some limits of the Linear Method and argue why it is essential to use copies and apply partitions as introduced in the previous Subsections. To exemplify this let us consider the following weighting function $w'(v_i) := V \to \mathbb{R}$ for nodes with $n = |V|$

$$w'_{v_i} := w'(v_i) = \begin{cases} 1 & , i = 1 \\ \frac{1}{1-n} & , \text{else} \end{cases}$$

and assume each node owns only one position $r_v \in M$. First one can see that the desired share for node v_1 is $\frac{1}{2}$, if we use w' as weight distribution. Than it turns out that using the Linear Method v_1 will receive more than his fair share, still under the condition that only one position for each node is chosen.

Theorem 6. *Without copies the Linear Method for n nodes with weights $w_{v_1} = 1$ and $w_{v_2}, \ldots, w_{v_{n-1}} = \frac{1}{n-1}$ assigns a data element with probability $p(v_1) = 1 - e^{-1} \approx 0.632$ to node v_1, if n tends to infinity.*

Proof. First we show that the weight distribution leads to a wanted share of $\frac{1}{2}$. So v_1 should expect $\frac{1}{2} \cdot |D|$ of the items we distribute among the nodes:

$$\begin{aligned} p(v_1) &= \frac{w_{v_1}}{\sum_{i=1}^{n} w_{v_i}} \\ &= \frac{1}{1 + \sum_{i=2}^{n} \frac{1}{n-1}} = \frac{1}{2} \end{aligned}$$

Now we can use height distribution of Lemma 3 and reduce the probability to the following term if n tends to infinity

$$\begin{aligned} \lim_{n\to\infty} \int_{x=0}^{1} x \left(1 - \frac{x}{n-1}\right)^{n-1} dx &= \int_{x=0}^{1} x e^{-x} dx \\ &= \left[-e^{-x}\right]_0^1 \\ &= 1 - e^{-1} \approx 0.632 \end{aligned}$$

□

This shows that the Linear Method needs copies to improve the balancing qualities. It is not enough to introduce partitions, although this leverages the size of intervals. So, the number of copies always improves the balancing quality of the Linear Method.

Concluding till here, we have seen in previous Subsections, with a sufficient constant number of virtual nodes and by coevally applying partitions from Section 2.2.2, we can improve the balancing bounds of $DHHT_{Lin}$ up to a constant factor. Based upon Theroem 5 and the monotone behavior, we know that in case of dynamical changes in the environment the insertion or removal of a node is $(1+\epsilon)$-competitive with high probability.

Corollary 8. *In a dynamical environment $(V, D, f_{DHHT_{Lin}})$, where V consists of heterogeneous nodes and $|W| = 1$, the $DHHT_{Lin}$ strategy is space balanced by using $\lceil \frac{2}{\epsilon} + 1 \rceil$ and additional $\mathcal{O}(log|V|)$ partitions. Further, $DHHT_{Lin}$ is adaptive and monotone with an $(1+\epsilon)$ competitiveness for any $\epsilon > 0$ if nodes are added or removed from the environment. So it solves the heterogeneous problem.*

In the following we try to reduce the additional overhead of $DHHT_{Lin}$, by substituting the linear scaled distance function with a logarithmic scaled distance function. This modification should improve the basic balancing quality such that the number of needed copies can be reduced.

2.2.3 The Logarithmic Method

In this section we present the second DHHT variant, the *Logarithmic Method*, in the following abbreviated with $DHHT_{Log}$. It is designed for resource allocations in dynamic environments $(V, D, f_{DHHT_{Log}})$ and resolves heterogeneity of nodes according to a single attribute, so $|W| = 1$.

The aim of $DHHT_{Log}$ is to improve the balancing quality of $DHHT_{Lin}$ and coevally reduce its managing overhead. The Logarithmic Method uses basically the same techniques to assign items to appropriate nodes, but it uses logarithmic scaled distance functions to respect the node weights. The example in Figure 2.10 shows the modified model and the mapping among nodes with such distance function. One can see there, that the hash range range is again divided into several responsibility subranges, but they differ in construction compared to Figure 2.9. One effect which can be observed directly is that pairwise different distance functions can possibly intersect multiple times. In some cases this will result into diverse responsibility ranges and thus different item assignments. The results in following subsections will show that this scheme also provides a fairly balanced distribution for data elements and is also capable of compensating dynamics with

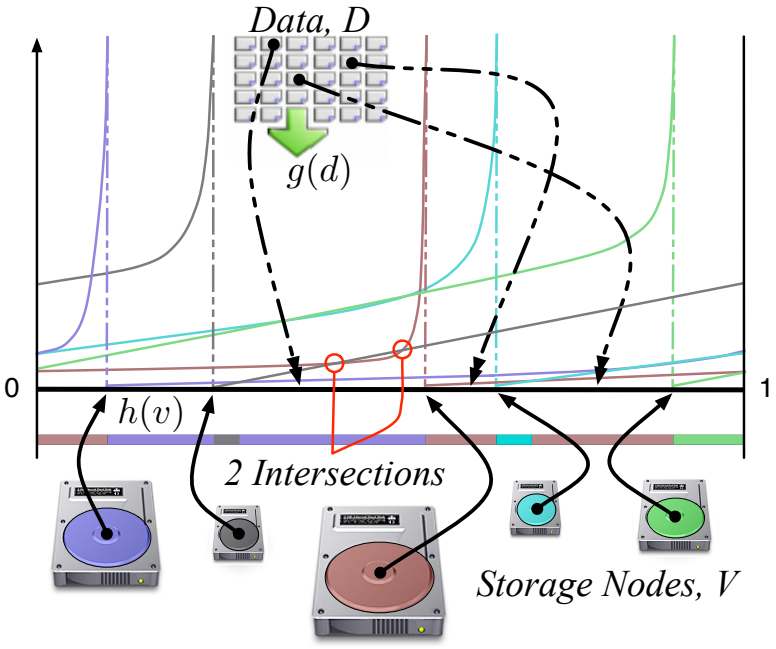


Figure 2.10: DHHT, the Logarithmic Method

a minimal effort, concerning data movement in case of node insertions, removals or system reorganization.

The Basic Method

As before, given a dynamic environment $(V, D, f_{DHHT_{Log}})$ consisting of a node set $V = \{v_1, \ldots, v_n\}$ and heterogeneous according to one attribute. The node weights are given by $W_{DHHT} := \{w(v_i) \in \mathbb{R}^+\}$, reflecting the dissimilarity of V, similar to 2.2.2. Further, we have a uniform item set $D = \{d_1, \ldots, d_m\}$ we want to distribute. The mapping function $f_{DHHT_{Log}}$ determines the rules for an item mapping. To map nodes into M we use the hash functions $r_v := h(v)$, $v \in V$ and $r_d := g(d)$, $d \in D$ to map items. Further, we still assume that both hash functions behave like independent uniform random variables in the hash range.

As mentioned, the mapping is technically the same, but instead of a using linear scaled distance function we use a logarithmic scaled distance function $l(x) = -ln(1-x)$. Further on we use only one function per node to represent its weight and the hash range M is still interpreted as unit ring. Thus, the height an item receives on a certain node $v \in V$ is determined by these two functions

$$D_{DHHT_{Log}}(r_d, r_v, w_v) := \begin{cases} \frac{-\ln(1-(r_d-r_v))}{w_v} & \text{if } r_d \geq r_v \\ \frac{-\ln(1-(1+r_d-r_v))}{w_v} = \frac{-\ln(r_v-r_d)}{w_v} & \text{else} \end{cases}$$

both functions can be transformed by applying Definition 8 to the following closed form

$$D_{DHHT_{Log}}(r_d, r_v, w_v) = \frac{-\ln(1-(r_d-s_v) \bmod 1)}{w_v}$$

As specified in $DHHT_{Lin}$, $D_{DHHT_{Log}}$ returns the minimal positive value at position $r_d \in M$ for a node v hashed to position r_v with weight w_v. Again, for each node v and data element d we call the value of this function the height an item receives at node v. Accordingly the *Logarithmic Method* assigns an item d at r_d to a v if it minimizes the height over all nodes. Therefore, we define the following function which determines the minimal height according to any position $r \in M$ over all nodes, which coevally identifies the responsible node for a given r_d.

$$H_{DHHT_{Log}}(r, V) := \min\Big(\{D_{DHHT_{Log}}(r, r_v, w_v) \mid v \in V\}\Big)$$

The operations resulting from the height function implementing node insertions, node removals and item assignments are nearly identically to the operations

listed in Section 2.2.2. Likewise $DHHT_{Lin}$, each node v possibly owns multiple ranges of the lower envelope I_{min} so $I_v := \{I_v^i \mid I_v^i \in I_{min}\}$. This leads to the following operations

1. $insertItem_{DHHT_{Log}}(d)$: determine r_d and find the range $I_v^i \in I_{min}$, where $r_d \in I_v^i$. Then store d on the v, because $D_{DHHT_{Log}}(r_d, r_v, w_v) = H_{DHHT_{Log}}(r_d, V)$.

2. $insertNode_{DHHT_{Log}}(v)$: determine r_v and use $D_{DHHT_{Log}}(x, r_v, w_v)$ to identify intersections $P(u,v) \mid u \in V$, where $P(u,v).x$ in $I_u^i \in I_{min}$ and $P(u,v).y$. Adapt the intersected intervals and remove all completely dominated. Move all items previously assigned by removed intervals or by the changed fraction of intervals to the new node v.

3. $deleteNode_{DHHT_{Log}}(v)$: $\forall I_v^i \in I_v \cap I_{min}$ determine the lower envelope between $[I_v^i.start, I_v^i.end]$. Insert all new intervals to I_{min} and resize the previously intersected by $D_{DHHT_{Log}}(x, r_v, w_v)$. Remove $I_v^i \in I_v \cap I_{min}$ from I_{min}. Move all $d \in f^{-1}(v)$ to the corresponding intervals of the changed areas in I_{min}.

As before in $DHHT_{Lin}$, the operations needed to delete an item are equal to inserting one, but instead of storing the item it is simply removed from the node. Finally, one can see that only the calculation of the minimum is substituted. Since the substituted distance function is a strict monotonically increasing function we can argue similar to $DHHT_{Lin}$ if we consider the characteristics of item movements in case of dynamics. The operation $insertNode_{DHHT_{Log}}(v)$ has only an impact on those nodes responsible for a removed or changed interval which was or is part of the lower envelope. Each involved node moves items only to the new node v, which carries over their ranges and takes part at the lower envelope. So no item movements occur among other nodes. The same behavior we can observe for $deleteNode_{DHHT_{Log}}(v)$. A the leaving node only moves items to nodes covering the abandoned ranges of the lower envelope, previously belonging to the leaving node. With this, we derive the first property according to Definition 16, given in the following corollary.

Corollary 9. *The Logarithmic Method of DHHT, $DHHT_{Log}$ is monotone concerning insert or delete operations of nodes.*

Again, this monotone behavior does not depend on the targeted balancing attitude of such a heterogeneous node representation. However, the representation of a node never changes and it is independent on the number of leaving or joining nodes and their weights. The aim of this model is to obtain the heterogeneous item distribution indirectly. Remember, till here we do not know if

this node representation reflects the objected weight distribution or improves the Linear Method.

In the following we proof the balancing quality of the $DHHT_{Log}$ mapping scheme. Therefore we take a look at the distribution of the height h according to a position $r \in M$ and define this in the following lemma

Lemma 5. *Given n nodes with weights $w_1, \ldots, w_n$. Then the height $H_{DHHT_{Log}}(r)$ assigned to a position r in M is distributed as follows:*

$$\mathbf{P}[H_{DHHT_{Log}}(r) > h] = e^{-\sum_{i \in V} w_i h}$$

We can observe that for small h this probability is close to the corresponding probability of the Linear Method. Furthermore, the probability $\mathbf{P}[H_{DHHT_{Lin}}(r) > \ell]$ of the Linear Method tends also for larger h to this probability if we increase the number of copies, denoted with c

$$\begin{aligned} \lim_{c \to \infty} \mathbf{P}[H_{DHHT_{Lin}}(r) > \ell]^{1/c} &= e^{-\sum_{i \in V} w_i h} \\ &= \mathbf{P}[H_{DHHT_{Log}}(r) > h] \end{aligned}$$

We know that the Linear Method is fair, if the number of copies grows towards an infinite number. This gives us an alternative proof for the following theorem

Theorem 7. *In a dynamic environment $(V, D, f_{DHHT_{Log}})$ consisting of heterogeneous nodes with $|W| = 1$ and positive weights $w_i := w(v_i)$, the Logarithmic Method assigns an item $d \in D$ to a node $v_i \in V$ with the probability of*

$$\mathbf{P}[d \text{ assigned to } v_i] = \frac{w_i}{\sum_{j=1}^{n} w_j}$$

Proof. Hence the probability that an item receives height $H_i := H_{DHHT_{Log}}(r_d)$ in the interval $[h - \delta, h]$ and receives larger height than h is defined by the condition $con_1 := (H_i \geq h - \delta \wedge H_i < h \wedge \bigwedge_{j \neq i} H_i \geq h)$ and at most

$$\begin{aligned} \mathbf{P}[con_1] &= \left(e^{-w_i(h-\delta)} - e^{-w_i h}\right) \prod_{j \neq i} e^{-w_j h} \\ &= e^{-w_i h} \left(e^{w_i \delta} - 1\right) \prod_{j \neq i} e^{-w_j h} \\ &= \left(e^{w_i \delta} - 1\right) \prod_{j \in [n]} e^{-w_j h} \end{aligned}$$

With this we show the upper-bound of the probability that an element is assigned to node v_i. This given by the following sum

$$\begin{aligned}\lim_{\delta\to 0}\sum_{k=1}^{\infty}\left(e^{w_i k\delta}-1\right)\prod_{j\in[n]}e^{-w_j\delta k} &\geq \lim_{\delta\to 0}\sum_{k=1}^{\infty}k\delta w_i e^{-\sum_{j\in[n]}w_j\delta k}\\ &= \int_{x=0}^{\infty}xw_i e^{-\sum_{j\in[n]}w_j x}dx\\ &= \frac{w_i}{\sum_{j\in[n]}w_j}\end{aligned}$$

A sufficient condition that a data element receives height in the interval $[h, h+\delta]$ and receives larger height than h is defined by $con_2 := (H_i \geq h \wedge H_i < h+\delta \wedge \bigwedge_{j\neq i} H_i \geq h)$ and the probability therefor is

$$\begin{aligned}\mathbf{P}[con_2] &= \left(e^{-w_i(h)}-e^{-w_i(h+\delta)}\right)\prod_{j\neq i}e^{-w_j h}\\ &= e^{-w_i h}\left(1-e^{-w_i\delta}\right)\prod_{j\neq i}e^{-w_j h}\\ &= \left(1-e^{-w_i\delta}\right)\prod_{j\in[n]}e^{-w_j h}\end{aligned}$$

With this we show that the probability of an element to be assigned to node v_i is lower-bounded by the following sum

$$\begin{aligned}\lim_{\delta\to 0}\sum_{k=1}^{\infty}\left(1-e^{-w_i k\delta}\right)\prod_{j\in[n]}e^{-w_j\delta k} &\leq \lim_{\delta\to 0}\sum_{k=1}^{\infty}k\delta w_i e^{-\sum_{j\in[n]}w_j\delta k}\\ &= \int_{x=0}^{\infty}xw_i e^{-\sum_{j\in[n]}w_j x}dx\\ &= \frac{w_i}{\sum_{j\in[n]}w_j}\end{aligned}$$

Both bounds proof the statement of the theorem □

As already mentioned to Theorem 3, the statement in Theorem 7 holds for n nodes and the placement of one data element inserted at the same time. In this situation, we achieve a perfect balance in expectation. If we insert more items at the same time, then we face strong dependencies between the assignments of the data elements. For the second element the probability that this element is inserted at some node v_i is highly dependent on whether the first element has been inserted at this node. This follows simply by the fact, that the intervals are fixed and their sizes determine the probability distributions.

The Use of Partitions

The result of Section 2.2.2 imply that we have similar effects to cope within the Logarithmic Method. We also use random position sampling to settle nodes in M, and once assigned, r_v never changes. Finally, as long as no dynamics occur in V the lower envelope and its range assignment to nodes is fixed. Further, in the balanced case, where $\forall v \in V \; |I_v| = \frac{w_v}{W_{all}}$ the Coupon Collector Effect occurs, see 1.3. Eventually, some nodes will receive intervals larger than desired, which directly results into an unavoidable imbalance. However, we can overcome this problem by using partitions again.

Therefore, we partition the hash range M into $\mathcal{O}(\log n)$ consecutive subranges of equal size. For each of these partitions we apply the mapping of $DHHT_{Log}$ individually. So all nodes $V = \{v_1, \ldots, v_n\}$ are mapped to each partition $M_1, \ldots, M_k$, with $M_i = [(i-1)/k, i/k)$ and $k = \mathcal{O}(\log n)$. Finally, the items are distributed among the partitions and within M_i assigned to the node which is the closest in this sub-range according to the logarithmic weighted height function.

Theorem 8. *For all $\epsilon > 0$ and $c > 0$ there exists $c' > 0$, where we apply the Logarithmic Method with $c' \log n$ partitions. Then, the following holds with high probability, i.e. $1 - n^{-c}$. Every node $v_i \in V$ receives data elements with probability p_i such that*

$$(1-\epsilon) \cdot \frac{w_i}{W} \;\leq\; p_i \;\leq\; (1+\epsilon) \cdot \frac{w_i}{W} \,.$$

Proof. This result follows by Theorem 7 and applying Chernoff Bounds from Theorem 1. □

The advantage of partitions are the same as before in $DHHT_{Lin}$, see Subsections in 2.2.2. In contrast to using a single hash range, partitions are independent multiple instances of the Logarithmic Method. Copies of the same node within a single has range possibly compete against each others, but they can not interact among partitions. This independency will compensates such bad samples of node and item positions.

Nevertheless, the restrictions considering the number of partitions and monotonicity are still existing. If the monotonicity must be guaranteed all the time, the partition number should not de- or increases continuously. The monotonicity is endangered, because partition borders need to be adjusted. These adjustments will produce re-injection of nodes, which is followed by item reassignments. However, since we use $\mathcal{O}(\log n)$ partitions we can keep the monotonicity by using Assumption 1 or Assumption 2, where we set the number of partitions to $\mathcal{O}(\log N)$.

Corollary 10. *In a dynamical environment $(V, D, f_{DHHT_{Log}})$, where V consists of heterogeneous nodes and $|W| = 1$, the $DHHT_{Log}$ strategy is space balanced by using additional $\mathcal{O}(log|V|)$ partitions. Further, $DHHT_{Log}$ is adaptive and monotone with an $(1+\epsilon)$ competitiveness for any $\epsilon > 0$ if nodes are added or removed from the environment. So it solves the heterogeneous problem.*

2.3 Advanced Techniques and Properties of DHHT

In this section we take a look on further techniques, to emphasize the practicability and expose some specific properties of DHHT. The introduced techniques will be applicable for both DHHT variants, so we consider them in general. The features we are describing are interesting for some application areas, like SAN or P2P, where network bandwidth, high data availability and efficient meta data information management is an issue.

2.3.1 Fragmentation

Fragmentation occurs in computer storage and is a kind of inefficiency, which possibly leads to reduced storage capacity, additional management overhead, all resulting in decreased access performance. The reason for tolerating such disadvantages is founded by the aim to keep allocation algorithms simple. The resulting fragmentation can be classified in the following groups:

- ***Data Fragmentation***, this effect occurs whenever the allocation of memory for a data element can not be done consecutively within the storage. This means, if, the item needs to be split into several peaces to be stored.
- ***External Fragmentation***, this effect occurs if free storage is split into several peaces and is not available in one peace. High External Fragmentation leads often to higher Data Fragmentation.
- ***Internal Fragmentation***, this effect occurs if memory is allocated, but without the intention to be used completely. This is often done for simplification in file systems. For instance, the file system has a block size k and the file needs to allocate $c(d)$ of memory. Typically, there will be $\lceil c(d)/k \rceil$ used, with a waste of $\lceil c(d)/k \rceil - c(d)/k$ of memory to keep the allocation of other files simple.

In case of DHHT we have to define what fragmentation means. We distribute homogeneous items which can be used as a consecutive number of physical blocks. Since all items are of same size, we typically inherit similar effects filesystems

have. The maximum number of consecutive physical blocks on a node is predefined by $c(d)$. If it is more, it was generated by random. Further, the allocation size $c(d)$, if $c(v)/c(d) \in \mathbb{N}$ leads to no additional capacity waste, otherwise if $c(v)/c(d) \in \mathbb{R}$ the waste is at most $\sum_{v \in V}(c(v)/c(d) - \lceil c(v)/c(d) \rceil)$ for all nodes.

Beside these classical effects, DHT like approaches are using a hash range as mapping universe, so DHHT does. Obviously the range fragmentation increases with the number of nodes and virtual nodes. This implicates that the number of responsibility subranges increases and therewith the node localization for item assignments. For example, to achieve space balance a DHT based schemes needs $\mathcal{O}(\log n)$ virtual injection points for each node. Thus the uniform DHT model has a hash range fragmentation of $\mathcal{O}(\log n \cdot n)$. Thus a balanced binary search structure sorted among r_v would need $c(\log(c' \cdot \log n \cdot n)$ steps. So it seems to be reasonable to keep the hash range fragmentation as small as possible.

As we have seen in Section 2.2.2 and 2.2.3 our DHHT approach generates additional fragments, caused by intersections of the distance function. The following theorems describe this fragmentation behavior. So we count the number of intervals a data element can be possibly assigned to.

Theorem 9. *The Linear Method with q copies and k partitions has a fragmentation of at most* $2qkn - 1$ *for n nodes. The Logarithmic Method using k partitions has a fragmentation of at most* $2kn - 1$.

Proof. This theorem follows by using results from the geometric applications of Davenport Schinzel Sequences [2] in the context of creating lower envelopes of totally and partially defined functions. Further we can combine these results with the observation in case of partially defined functions that our model and its distance functions have all the same end point and beside their injections the same entry point. □

From our previous results we can derive a standard choice of parameters for $q = \mathcal{O}(1)$ and $k = \mathcal{O}(\log n)$ to achieve constant precision with high probability.

Corollary 11. *The DHHT scheme provides a fragmentation of* $\mathcal{O}(n \log n)$*, if we use a constant number of node copies and* $\mathcal{O}(\log n)$ *partitions.*

A final note on this logarithmic overhead. We have to compare this fragmentation with the fragmentation produced if we use the Normalization, described in Section 2.1.5. The complexity obtained by resolving heterogeneity with an uniform strategy depends on the heterogeneity distribution. The higher the ration w_{max}/w_{min} the more injection points are needed, whereas our heterogeneous model is independent on this ratio.

2.3.2 Node Fading and Data Migration Prediction

In some application areas it is of much importance that in case of dynamics the allocation of further involved resources like network bandwidth is well controlled. The aim is to avoid peak load caused by maintenance operations after node deletions, insertions or capacity changes, because they might decrease the response time of other user requests. If such changes are known in advance, they can be handled with protocols coordinating the migration among nodes. If changes happen unexpected, redundant data encodings, like Reed Solomon [35], are used for compensation. However, there is a tradeoff, short term migration allocates high bandwidth, whereas long term migration allocates additional memory to keep the data context.

In the following we introduce two DHHT attributes, both helpful to define efficient strategies or protocols to migrate items among nodes. The attributes are obtained from the independent and self-scaled representation of node heterogeneity in DHHT and on the basis of the guaranteed hash range coverage by the distance functions

Node Fading

The first feature we point out is *Node Fading*, which can be applied in case of controlled dynamics, where environment changes are known in advance. In the fol-

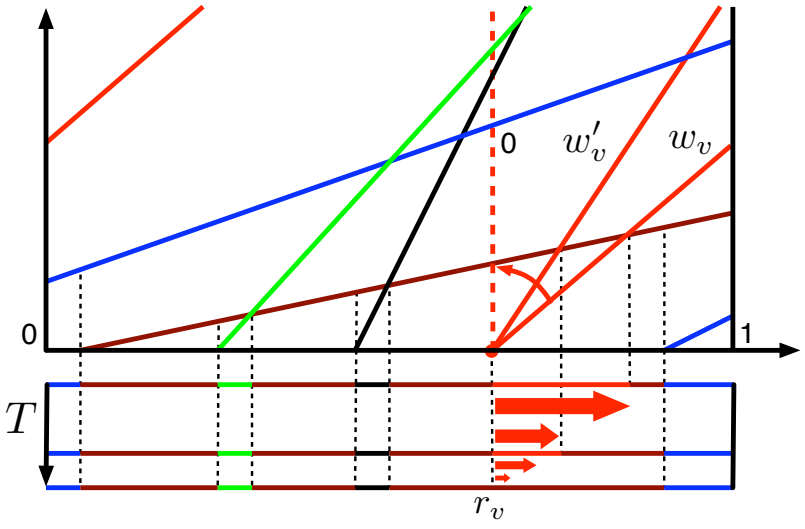


Figure 2.11: DHHT and Node Out Fading

lowing we consider the capacity change of a node, because insertion or removal are only special cases of changes. Typically, such changes and the necessary maintenance operations are not directly respected within distribution strategies. The overall consumption of involved resources is dominated by the competitiveness and especially the monotonicity of the method.

The idea of fading is to change $v \in V$ smoothly over time from $w_v \xrightarrow{k} w'_v$ controlled by an arbitrarily large, but finite number of k steps and migrate items by disposing embedded information and parameters of DHHT. As we have seen, both DHHT variants use strictly monotonic increasing functions to represent a node v. Thus, each $d \in f^{-1}(v)$ receives a different height, because $\forall\, d, d' \in f^{-1}(v)$ $r_d \neq r_{d'}$, if $d \neq d'$. Since v stores d it must define a subrange $I_v \in I_{min}$ with $r_d \in I_v$. In the following we only consider the intervals of I_{min}.

So, let us assume we decrease the w_v of v, then each time we set w_v a step closer to w'_v. Thus, according to the method, the gradient increases and the height of all $d \in f^{-1}(v)$, whereas competing heights on other nodes remain the same. Regarding the lower envelope, we can observe with each step the decrement of $|I_v^i|$ and the following geometric behavior:

1. if I_v is defined by $[r_v, P(v,u).x]$, than $P(v,u).x$ shifts towards r_v, where u is the adjacent node defining the next range of I_{min}.

2. if $I_v := [P(v,u).x, P(v,u').x]$ is defined by intersections, $P(v,u).y$ increases and $P(v,u).x$ shifts apart r_v, whereas $P(v,u').y$ decreases and $P(v,u').x$ shifts towards r_v. Here u and u' are part of the complete range adjacencies of I_v within I_{min}. Note, in case of $DHHT_{Log}$ $u = u'$ is possible.

From this we can conclude, according to the mapping rules of Section 2.2.2 and 2.2.3, that the items $f^{-1}(v)$ will migrate among a descending or among a descending and ascending order of their heights and not arbitrarily. If a node is responsible for many subranges of the lower envelope, the ordered migration occurs for each subrange $I_v^i \in I_{min}$ individually. Clearly, in case of increasing w_i the ranges are increasing and the items will migrate among a sorted order from the other nodes to v.

As mentioned only w_v changes, other node weights are left untouched and both variants use strictly monotonic increasing functions on independently and randomly selected positions. So, based on Corollary 7 and Corollary 9 it follows, that, if we change w_v to w'_v, there will be no item movements among other nodes, which leads to the following corollary

Corollary 12. *Let w_v be the current weight of $v \in V$, than fading $w_v \xrightarrow{k} w'_v \mid k \in \mathbb{N}^+$ in DHHT is monotone.*

Essentially, fading is the same as removing v with w_v and insert it again at the previously used r_d with w'_v. In both cases $f^{-1}(v)$ is equal after these operation.

A protocol handling controlled dynamics by using the observed attributes is now derivable. Let us assume that all $d \in f^{-1}(v)$ are sorted among their heights and associated with I_v^i. We denote with $I_v^{i\prime}$ the intervals belonging to v at r_v with weight w'_v. The main migration steps if we reduce $w_v \xrightarrow{k} w'_v$ are described in the following:

1. for each I_v^i determine $I_v^{i\prime}$ and with it $I_{Ldiff} := [I_v^i.start, I_v^{i\prime}.start]$ and $I_{Rdiff} := [I_v^i.end, I_v^{i\prime}.end]$ and the resulting new lower envelope I'_{min}
2. determine the items groups $G_R := f^{-1}(I_{Rdiff})$ and $G_L := f^{-1}(I_{Ldiff})$
3. sort G_R and G_L among the item heights and partition them consecutively among the heights in k subgroups $\{G_L^1, \ldots, G_R^k\}$ and $\{G_L^1, \ldots, G_R^k\}$ with same cardinality
4. in each step i migrate items from G_L^i and G_R^{k+1-i} to the lower envelope I'_{min}

So, the allocation of further resources like bandwidth can now be well controlled by k with respect to $|w_i - w'_i|$ and $c(f^{-1}(I_{Rdiff} \cup I_{Ldiff}))$ and lastly whether the migration of items in G_i is done in parallel or sequentially. The protocol is analogue for increasing w_v, whereas the interaction of each I_v^i with I_{min} is the other way round. Note, 0 is a valid start or end weight, and by using an adequate k the total bandwidth used for inserting or removing nodes should not exceed a desired value.

Eventually, the reason why fading is efficiently to use is based upon two properties. The first is the self-scaling attitude, nodes are represented independently form each other and the relative sizes of responsibility ranges are obtained indirectly. The second is that insert and join operations of DHHT are monotone.

Migration Prediction

If nodes are inserted or changing their capacity and in worst case if they are failing, it would be convenient to know which data items are possibly involved in maintenance operations. So, the second property we like to point out is Migration Prediction. Remember, the distance functions of DHHT are assigning to each item a specific height. The height an item d receives from node v increases with $(1 + (r_d - r_v)) \bmod 1$, but the more nodes we insert, the less this height can be, because it is covered by other nodes. We have defined the height H_d of a data element d for finding its corresponding node. This height is proportional to the probability that this data element is moved, if a new node arrives.

Fact 1. *If in the basic Linear Method, without copies and without partitions, a node arrives with weight w_v, then the probability that data element d with previous height H_d is assigned to the new node is* $\min\{w_v H_d, 1\}$.

This fact follows by calculating the length of the interval describing possible positions where the new node receives this data element. In the Logarithmic Method the situation is similar:

Fact 2. *If in the Logarithmic Method, without copies and without partitions, a node arrives with weight w_v then the probability that data element d with previous height H_d is assigned to the new node is* $1 - e^{-w_v H_d}$

Again, we can see the relation between $DHHT_{Lin}$ and $DHHT_{Log}$ that for small values $H_d \ll \frac{1}{w_v}$, the term $1 - e^{-w_v H_d}$ can be approximated by $w_v H_d$.

So, concluding from both facts, the probability that a data element needs to be assigned to a new arriving node can be calculated in advance. Note that the order of these reassignment probabilities is independent from the weight of the arriving node. This feature can also be used for predicting the data migration of items. This allows further optimizing data storage for this purpose.

2.3.3 Efficent Data Structure

In this section we show that there exists an efficient data structure for both methods. Independent from the presented algorithms here, there are several opportunities to implement DHHT. Here we show that DHHT has a space complexity of $\mathcal{O}(n)$, a lookup time for items of $\mathcal{O}(\log n)$ and inserting or removing nodes needs the amortized time of $\mathcal{O}(1)$. Therefore, we use tables and some linked lists.

Theorem 10. *There is an algorithm that determines for a data element the corresponding node according the Linear Method in expected time $\mathcal{O}(\log n)$. The data structure has size $\mathcal{O}(n)$. Inserting and deleting nodes in this data structures needs amortized time $\mathcal{O}(1)$.*

Algorithm 1 InsertNode(v, T), inserting a new node v

Require: v, T
1: $s_v := h(v)$;
2: $T^* := InsertTable(T^*, s_v, v)$;
3: $\ell := \lfloor \log_2 w_v \rfloor$
4: $T_l := InsertTable(T_l, s_v, v)$;
5: **return** T;

Proof. As proof we present the algorithms for the data structure for the plain Linear Method without copies and partitions, see Algorithm 1, 2, 3, 4. The use of partitions can be considered as multiple instances of a single variant sharing a single hash range, whereas copies will only increase the number of nodes.

The nodes are classified into the sets $\ldots, V_{-2}, V_{-1}, V_0, V_1, V_2, \ldots$ according to their weights such that any node $v_i \in V_\ell \Leftrightarrow \lfloor \log_2 w_i \rfloor = \ell$. For each non-empty node-set we use a table $T_i[0, .., N_i - 1]$ of N_i elements, where N_i is chosen such that $N_i \leq |V_i| \leq 4N_i$. In this table we store at each entry $T_i[j]$ all nodes satisfying $s_v \in [j/N_i, (j+1)/N_i)$. These sets can be stored by a linked lists. Besides this, we provide a table T^* for all nodes organized as the other tables.

Lemma 6.

1. *For $V_i \neq \emptyset$ a set $T_i[j]$ is empty with probability of at most $3/4$. The probability that an interval $T_i[j, \ldots, j+d]$ consists only of empty sets is at most $(\frac{3}{4})^d$.*

2. *The expected number of elements in the sets $T_i[j], \ldots,$ $T_i[j+d]$ is at most $2d$ (even under the condition that the rightmost $d/2$ sets are empty).*

3. *If a data element x has d empty entries in T_ℓ left onwards from its position $p = \lfloor r_x \cdot N_\ell \rfloor$, i.e. $T_\ell[p-d+1], \ldots, T_\ell[p] = \emptyset$ and $T[p-d] \neq \emptyset$ then the corresponding node to x must lie in the in the sets $T_\ell[p-2d], \ldots, T_\ell[p-d]$ if it belongs to the layer ℓ.*

Now let $d = H2^{-\ell}$ then the expected number of nodes in this array is at most $2d$ and the chances to check so many entries is at most $(3/4)^d$. Summing over all d shows the expected running time to check one level T_i is constant. For the small weights $w_i \leq \frac{1}{n^2} \max_u \{w_u\}$ the probability to assign a data element to such a node is at most $\frac{1}{n^2}$. The probability that a data element is assigned to any such node is therefore at most $\frac{1}{n}$. Then, even linear time to detect this element leads to constant expected run time. □

Algorithm 2 DeleteNode(v, T), deleting a node v

Require: v, T
1: $s_v := h(v)$;
2: $T^* := DeleteTable(T^*, s_v, v)$;
3: $l := \lfloor \log_2 w_v \rfloor$;
4: $T_l := DeleteTable(T_l, s_v, v)$;
5: **return** T;

Algorithm 3 InsertTable(T, s, v), Insert a node v into a Table

Require: T, v, s, $N := size(T)$, $S := \sum_{i=0}^{N-1} |T[i]|$;
if (T is empty) **then**
  $N := 1$; $T[0] := \{v\}$;
else
  if $(N \leq S + 1)$ **then**
    $N := 2N$; $T'[0, \cdots, N-1] := \emptyset$;
    for all $v \in \bigcup_{i=0}^{N/2-1} T[i] \cup \{v\}$ **do**
      $T'[\lfloor s/N \rfloor] := T'[\lfloor s/N \rfloor] \cup \{v\}$;
    end for
    $T := T'$;
  else
    $T[\lfloor s/N \rfloor] := T[\lfloor s/N \rfloor] \cup \{v\}$
  end if
end if
return T;

Algorithm 4 DeleteTable(v, s_v, T), deleting a node v from a table

Require: T, v, s_v, $N := size(T)$, $S := \sum_{i=0}^{N-1} |T[i]| - 1$;
$T[\lfloor s/N \rfloor] := T[\lfloor s/N \rfloor] \setminus \{v\}$;
if $(S == 0)$ **then**
  $T := empty()$;
else
  if $(S < N/4)$ **then**
    $N := N/2$;
    for all $v \in \bigcup_{i=0}^{2N-1} T[i]$ **do**
      $T'[\lfloor s/N \rfloor] := T'[\lfloor s/N \rfloor] \cup \{v\}$;
    end for
    $T := T'$;
  end if
end if
return T;

As said before, we have included neither partitions nor copies. By introducing q copies and k partitions, the needed space for this data structure rises up to $\mathcal{O}(knq)$, while the running time is $\mathcal{O}(\log n + \log q)$ for data lookup and eventually $\mathcal{O}(qk)$ for inserting or deleting a node.

For the *Logarithmic Method* we can re-use the presented data structure of the

Algorithm 5 *LookUp*(T, d), finding appropriate node v for an item d

Require: $l_{max} := \max\{i : V_i \neq \emptyset\}$, $r_d := g(d)$, $n := |V|$;
1: $H := \infty$;
2: **while** $(l \geq l_{max} - 2\log n - 1)$ **do**
3: $p := \lfloor r_d \cdot N_l \rfloor$; $q := p$; $d := 0$;
4: **while** $(T_l[q] == \emptyset)$ **do**
5: $q := (q - 1) \mod N_l$; $d := d + 1$;
6: **end while**
7: **for** $(i := q - d - 1; i \leq q)$ **do**
8: **for all** $v \in T_l[i \mod N_l]$ **do**
9: $s_v := h(v)$;
10: **if** $(H > D_{w_v}(r_d, s_v))$ **then**
11: $y := v$; $H := D_{w_v}(r_d, s_v)$;
12: **end if**
13: **end for**
14: **end for**
15: **end while**

Linear Method.

Theorem 11. *There is an algorithm that determines for a data element the corresponding node according to the Logarithmic Method in expected time* $\mathcal{O}(\log n)$*. The data structure has size* $\mathcal{O}(n)$*. Inserting and deleting nodes in this data structures needs amortized time* $\mathcal{O}(1)$*.*

Proof. We can re-use the data structure presented for the Linear Method. We just replace all occurrences of the linear height function D_v by the logarithmic height function. One can show that Lemma 6 also holds for the Logarithmic Method as well as for the low weighted nodes, the same run time analysis is valid. □

2.3.4 Double Hashing

The idea of Double Hashing is an extension of DHHT by using multiple hash functions. The aim is to increase the node independence and to overcome the intrinsic imbalance obtained by fixed node positions and the resulting lower envelope. The enhancement is to increase the number of hash functions by the number of nodes. Each node $v_i \in V$ owns an individual hash function $h_i : M \to M$. With h_i a node v determines its weighted distance $r_{i,d}$ and thus the height H_i of d individually. Note, we still assume that all hash functions h_i behave like independent uniform random variables in the hash range M.

The necessary steps to assign one item d to its appropriate node, in difference to the original mapping scheme of Section 2.2.2 and Section 2.2.3, are described by the following steps:

1. determine for $d \in D$ the injection point r_d by using a globally know single hash function $g(d) : D \rightarrow M$
2. determine for each node $v_i \in V$ by using their individual hash functions $r_{i,d} := h_i(r_d) = h_i(g(d))$
3. Calculate the weighted heights $H_{*,v}$ of d for all $v_i \in V$, where $H_{Lin,v} := r_{v,d}/w_v$ according the Linear Method or $H_{Log,v} := -\ln(1 - r_{v,d})/w_v$ according to the Logarithmic Method
4. Assign d to v if $H_{*,v} = \min(\{H_{*,u} \mid u \in V\})$.

The scheme is exemplified in Figure 2.12 and shows the mapping decision for one item d among five nodes with positive weights $w_v \in \{w_1, \ldots, w_5\}$. The illustration uses the weighting of the Linear Method, if the Logarithmic Method should be used H_{Lin} must be substituted by H_{Log}.

Theorem 12. *The Linear Method using Double Hashing assigns data elements to all of the n nodes with probabilities p_i for each node, such that*

$$(1 - \sqrt{\epsilon}) \cdot \frac{w_i}{W} \leq p_i \leq (1 + \epsilon) \cdot \frac{w_i}{W}.$$

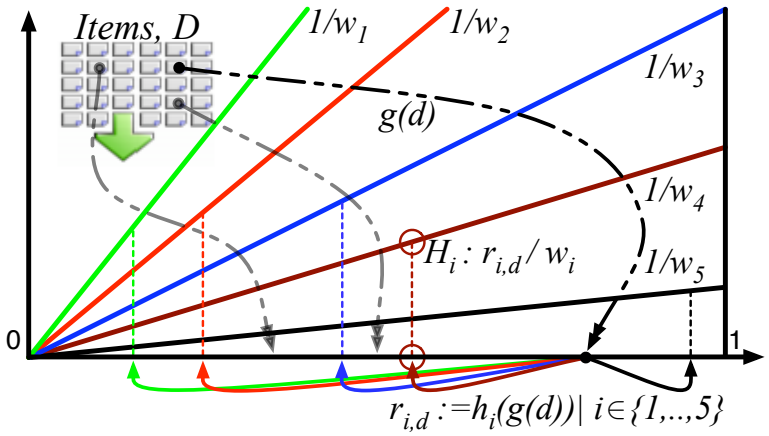


Figure 2.12: Double Hashing and $DHHT_{Lin}$

The Logarithmic Method using Double Hashing assigns data elements to all of the n nodes with probabilities p_v for each node, such that

$$p_v = \frac{w_v}{W}$$

Proof. This can be concluded by the proof of Theorem 4 according to the Linear Method and Theorem 7 according to the Logarithmic Method. Further, we use for both variants the fact that now the heights of items are independently distributed. □

The main advantage of this extension is its achievement of the same probability distribution as the Linear Method or respectively the Logarithmic Method, if both are using a large number of partitions. The main drawback is the intrinsic linear running time to determine the appropriate node to place or find a data element, which results into $\mathcal{O}(n)$. Each item assignment must be calculated individually and can not be concluded from I_{min}. Especially in case of dynamics this mapping scheme has a negative effect, if a node is added, removed or changes its weight. The previously given lower envelope is missing and we cannot read off item movements efficiently, compare Section 2.3.2. So the steps needed to preserve the distribution and the data integrity are the following:

- *Insert Node v:* Determine the hash function h_v, move all items $d \in D$ to v, where $H_v(d) < H_u(d)$. This means each item needs to be touched and compared with the new height, which is similar to the pure random model in 2.1.3.

- *Weight Change v:* We distinguish two cases. First, if w_v decreases, than determine $H_{*,v}(d)$ and compare for all $d \in f^{-1}(v)$ the newly received height with all other nodes. If there exists a node u with $H_{*,u}(d) < H_{*,v}(d)$, move d to u which minimizes $H_{*,u}(d)$. Second, if w_v increases, determine for all $d \in \bigcup_{u \in V \setminus \{v\}} f^{-1}(u)$ the height $H_{*,v}(d)$ and move d to v if $H_{*,v} = \min(\{H_{*,u} \mid u \in V\})$.

- *Delete Node v:* Determine for each item $d \in f^{-1}(v)$ the node $u \in V$, where $H_{*,u}(d) < H_{*,k}(d)\ \forall k \in V \setminus \{v\}$. So, for each $d \in f^{-1}(v)$ we have to evaluate $n-1$ hash functions to determine the successor of v.

So, Double Hashing is reminiscent to the introduced Balls into Bin Model of Section 2.1.3, we need a lot of communication or many calculations, depending on the capacity of each node. This is practicable in parallel, but we can still avoid unnecessary expansive item movements and preserve the monotonicity of the scheme. To compensate some of these runtimes it might be helpful to store with

each item a constant number of successors. Nevertheless, this list must be kept up to date, so the length should reflect the number of expected changes to amortize the effort over time.

Finally Double Hashing is only reasonable if the number of nodes is comparatively small and the fragmentation of M is not an issue. Further, because of the costs in case of dynamics, insertions, deletions or weight changes should not occur frequently. If this can be agreed, the *Double Hashing* extension is an interesting choice for the Linear Method or the Logarithmic Method.

2.3.5 Other Distance Functions and Range Spaces

We have proposed the linear and the logarithmic height function in an one-dimensional ring to achieve a weighted version of distributed hash tables. One might ask which influence it has if we use different height measures or if we use two-, three- or higher-dimensional spaces. Therefore, we shortly discuss the following modifications. It is a straight-forward observation that a polynomial height function for some $\beta > 0$ we have

$$D_w(r,s) := \left(\frac{r-s}{w} \bmod 1\right)^\beta$$

as a replacement for the linear height function does not change anything. This follows from the fact that for determining the minimum height we apply the minimum function and the outcome remains the same. Of course any other monotonic growing function applied to $\frac{r-s}{w} \bmod 1$ leads to the same situation.

On first sight the situation seems to be more interesting if we map data elements and nodes to a two-dimensional space $[0,1]^2$ and use a distance function. However, one can reduce the probability distribution of a node receiving a certain height to a quadratic height function in one-dimensional space. So, for the balance we end up with the Linear Method in the one-dimensional space $[0,1)$. Analogously, it turns out that for every constant dimensional space and distance measure according to the probability distribution the situation is more or less the same as in the Linear Method.

2.3.6 Implications on the Heterogeneous Distribution Problem

The main challenge of any dynamic distribution scheme is to find a good and simple representation of heterogeneity, as we have seen in related approaches and our model. The probability of a node to receive an item should reflect its share in the environment at any time. If a hashing scheme is used, this is done via

segmentation of the hash range M in such way that the sum of segments a node owns or is jointly responsible for is equivalent in size of the desired expectation. If this is reached, we can assume that a weighted distribution is achieved if the items are distributed evenly over the hash range. Finally, the distribution quality only depends on the even item distribution among M or additional embedded mechanisms like multiple choice to compensate imbalance possibly obtained by random item distribution.

As long as we accept dynamic environment changes, the number of participating nodes can change permanently and the capacity of joining nodes is undetermined in size. If the configuration of V changes, the strategy reacts and afterwards the shares must be reflected again. Reacting means we can repair the desired distribution by remapping items, because of adapted responsibilities in M. Hopefully, the remapping is in the magnitude of the relative capacity change of C_V. Afterwards, we can start over with the item distribution. For some characteristics of these maintenance operations we already have measures to evaluate their efficiency and summarized them in distribution problems, see Definition 15 and Definition 16.

However, if the relative changes in V or W are significantly high, some models have to adjust the representation of the nodes within the model. This is needed to guarantee the continuing ability to overcome heterogeneity. For example, if the changes are beyond a specific range, the interval lengths of Share must be adapted. This is, because the length of an interval depends on the relative share of a node. As long as the relative change of C_V is small nothing happens. Typically, the smaller a node is the less is its impact on changes. For instance, Share does this only if the diversity of the current share of a node and the representation within the model reaches a limit, compare Section 2.1.5. Such a slackness can be justified, because otherwise each representation adaption possibly results in expansive item movements. A further example, the fall back bin v_{fb} in Sieve. It leads to a modified representation of v_{fb}, because of its increased probability to receive unassigned items. This probability must be recalculated if v_{fb} gets exchanged.

As we have observed, some of these drawbacks are reversed by features of the DHHT scheme. The representation of nodes never changes, except the node itself changes. If other nodes are joining or leaving, the shares are obtained indirectly without changing the global system representation. Especially, if nodes are leaving, the coverage of M is always guaranteed. Furthermore, in distributed environments this ignorance of global knowledge is helpful for local decisions. For instance, if v leaves only the nodes covering its ranges afterwards must be known by v to decide the item migration.

This distinction to other aproaches allows us to raise the bar for the heteroge-

neous distribution problem. This leads to the following extended definition.

Definition 17. *Let (V, D, f_W) be a dynamical environment, where V consists of heterogeneous nodes and D of uniform items. A mapping function f_W solves the distributed heterogeneous problem, if the following terms are fulfilled.*

1. *The mapping of $d \in D$ via f_W is simple. This is the case if f_W uses as input V, W and d without the knowledge of $\bigcup_{v \in V} f_W^{-1}(v)$.*
2. *The mapping is space balanced or fair if for any $v, u \in V$ $\frac{w_v}{f_W^{-1}(v)} \approx \frac{w_u}{f_W^{-1}(u)}$*
3. *The mapping function f_W is adaptive and monotone for join, leave and change operations*
4. *For all $v \in V$ $f_W(d)$ is unique and represents v independently.*

The extended problem definition has additional requirements, explained in the following definitions

Definition 18. *Let (V, D, f_W) be a dynamical heterogeneous environment, a mapping function $f_W(d)$ is unique, if after any change in V $f_W(d)$ is injective.*

This should emphasize the requirement of distributed environments that the responsibilities are always well defined. This should be the case even during dynamics and without changing the representation of the nodes.

Definition 19. *Let (V, D, f_W) be a dynamical heterogeneous environment, a node representation is independent for any $v \in V$ if the mapping is monotone if w_v changes.*

This means, however an environment changes the representation of nodes, except the changing one are fixed. So a node can change its weight and increase its probability to receive items and coevally takes over responsibilities from existing nodes without changing the representation of others. The advantage of such an attribute is that no additional costs can arise from adaptations and thus, there is no advantage in suppressing them.

The advantage of a unique and independent representation can be shown in distributed environments. The item assignment is based upon the representation of nodes. So, if a change of a single node has an impact on other representations, the change needs to be broadcasted within the environment, because all other representations must be adapted. Whereas an independent representations only changes a single node and involves those nodes taking over or giving away item responsibilities. So, these changes get along with less communication. Furthermore, an asynchronous item assignments can be implemented easily, because the

assignment can be negotiated pairwise. In case of DHHT this negotiation is transitive, so if $H_{*,v}(d) < H_{*,u}(d)$ and $H_{*,k}(d) < H_{*,v}(d)$ than $H_{*,k}(d) < H_{*,u}(d)$. So, DHHT covers Definition 18, this can be explained easily and is summerized in the following theroem.

Theorem 13. *In a dynamical environment (V, D, f_{DHHT_*}), where V consists of heterogeneous nodes and $|W| = 1$, $DHHT_{Lin}$ and the $DHHT_{Log}$ strategy solves the distributed heterogeneous problem if nodes are added to, removed from or changed within the environment.*

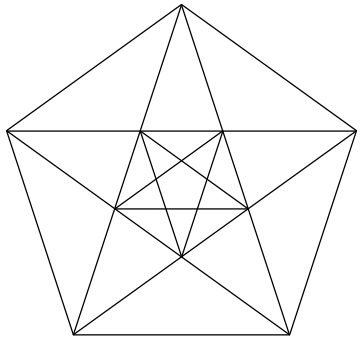
Proof. Since DHHT fulfills the restrictions of Definition 16, we only need to show the extension of Definition 3 and Definition 4. First the extension of Definition 3. As we have seen in Section 2.3.2 and pointed out in Corollary 12, the fading operation is monotone, too. Second the independent scaling of f_W. In both variants the mapping is decided by minimizing the distance to all nodes and the distance function for a node only depends on its absolute weight and not on the relative share of a node. The reflection of the share is indirectly reached by minimization of weighted distances. So, if v joins or leaves the environment, it has no impact on the distance values of other nodes. Further, since both variants use strictly monotonically increasing functions to represent a node, the required explicit coverage of M can be guaranteed. □

A final note on the previously introduced Double Hashing. If we consider the Double Hashing method using any of the DHHT variants as an own model, we can not attest the double hashing covering the extension of the distributed heterogeneous problem. Even in the uniform case we have declined the necessity of asking each item whether it wants to stay or not. As we have seen, this is the main drawback of Double Hashing, changing means touching each item. This should be avoided in large environments storing many items, definitely!

Chapter 3

Unit Ring Decomposition

As we have seen many DHT based data structures are using an unit range M. With hash functions, behaving like independently distributed random variables M gets decomposed it into several intervals. The intervals are assigned to nodes defining responsibility ranges for items distributed among M. In this Chapter we introduce a deterministic and greedy algorithm for unit ring decomposition. We will show the basic steps needed to overcome dynamical changes in V and its performance. The algorithms and the data structure are inspired by linear hashing using the golden ration. It is transformed to a greedy algorithm, useable in distributed and dynamic environments, handling node insertions, removals, and item assignments. The major aim is to underbid the naive deterministic halving approach. We will regard the smoothness, see Definition 23, of our algorithms and discuss improvements reached by some changes, see Section 3.2.1 and Section 3.2.2. The major aim of this is to underbid the naive deterministic halving approach with its smoothness of 2, see 3.1.2.



3.1 Unit Ring Decomposition

As we have seen in Section 2.1.3 a very popular technique to mange a set of nodes V is Consistent Hashing aka. DHT. It can be used for distributing any kind of work load D among V with the aim that each node receives and expects a workload of $|D|/|V|$ over time. Therefore, DHT and many other related data structures are using a unit range or unit ring as mapping universe to decide a mapping between D and V. Commonly a double sided hashing for nodes and item is used. Mainly the node hashing is the transition from the continuos to the discrete space, producing a fragmentation of the unit ring we denote in the following the decomposition of M. The main challenge is to avoid imbalances especially caused by changes in $|V|$ and to compensate them with as less effort as possible. In this chapter we do not consider how to spread items evenly within M, but we assume that the length of intervals a node is responsible for reflects its wanted probability to receive an item. To compare and evaluate strategies we only look at these lengths. First, we have to define the relation between the nodes, their ranges and the injection points of nodes.

Definition 20. *For $v, u \in V$ let $r_v, r_u \in M$ and $r_u = min(\{r_u \mid r_u > r_v\})$, then v owns $I := [r_v, r_u[$ and $|I| := r_u - r_v$ is its length. Further, I_v is the set of intervals a node $v \in V$ owns, $\#I_v$ its cardinality, $|I_v| := \sum_{I \in I_v} |I|$ and $\sum_{v \in V} |I_v| = 1$.*

In the following we will denote with v_L the left and with v_R the right adjacent node of any $v \in V$. Clearly, a fragmentation of M can be obtained in several ways. In this chapter we consider decompositions of M with specific attributes described by the following definition.

Definition 21. *We say $\bigcup_{v \in V}\{I_v\}$ is a $M_{c,\Lambda}$-decomposition or $M_{c,\Lambda}$-dc, if it is and generated by the algorithm Λ or a constant Λ and $1 \leq \max(\{\#I_v\}) \leq c$. Further, we denote a decomposition as strict, if $\forall\ I_v \in M_{c,\Lambda} : \#I_v = c$.*

So, the fragmentation of $M_{c,\Lambda}$ is at most $c \cdot |V|$. Since our goal is to assign to each node the same fraction of items and the length of a range reflects this, the following definition describes the optimum a decomposition can reach.

Definition 22. *A $M_{c,\Lambda}-$decomposition or strict $M_{c,\Lambda}-$dc is optimal, if for each v*

$$|I_{opt}| := 1/n = |V|^{-1} = |I_v|, \forall v \in V$$

is true.

To measure and compare the quality of any decomposition, we introduce the *smoothness* of a $M_{c,\Lambda}$-dc.

Definition 23. *Let $M_{c,\Lambda}$-dc a decomposition of M, then the smoothness of a $M_{c,\Lambda}$-dc denotes the ratio $s_{M_{c,\Lambda}} := max(\{|I_v|/|I_u| \mid v, u \in V\})$.*

The smoothness of $M_{c,\Lambda}$-dc is a really hard measure, because we will consider it with respect to dynamics, and look at the worst ratio at any point of time. Note, there is a trade off between smoothness and the fragmentation. We can obtain a better smoothness if we increase the number of intervals a node owns and consider only $|I_v|$. This increases the unwanted fragmentation of M and thus the managing overhead in many application areas. So, our main objection is a low smoothness with a low fragmentation coevally.

A further challenge we have to face is to avoid readjustments of node positions $r_v \in M$ during dynamics and to keep the smoothness sustainedly low. If r_v is determined once, it should never change again unless the node leaves the network. This is motivated by the application area SAN. Such a restriction is important, because adjusting r_v leads to costly item movements and to possible non monotone behavior. In contrast, peers in Peer to Peer networks can use leave and join operation to rebalance their expected workload. However, these self adjusting leave and rejoin operations are heavily expansive in SANs.

To exemplify such costs let us consider the following simple example. Assuming the strict $M_{1,\Lambda}$-dc with $|V| = n$ is optimal and for all $d \in D$ the according r_d are evenly spread among M. If we add a node u, so $|V| = n + 1$, and have to keep $\#I_v = 1$ and $s_{M_{1,\Lambda}} = 1$, then we must allow readjustments of r_v. Otherwise it will not be possible to insert u and rebalance afterwards. However, these position shifts will lead to a chain reaction of item movements. So, if we assign the range $I_u := [1 - \frac{1}{n+1}, 1[$ to u it receives $\frac{n}{n+1} \cdot f^{-1}(u_L)$ from its adjacent node u_L, previously responsible of $I_{u_L} := [1 - \frac{1}{n}, 1[$. Afterwards, u_L itself needs to readjust too and so on for every node, until each is responsible for a range of size $|I_v| = \frac{1}{n+1}$. Since with each readjust an according fraction of $f^{-1}(V)$ is transfered, this results in the following readjusts and item movements of all nodes

$$f^{-1}(V) \cdot \sum_{i=1}^{n} \frac{i}{n(n+1)} = f^{-1}(V) \cdot \frac{n(n+1)}{2n(n+1)} = f^{-1}(V) \cdot \frac{1}{2}$$

We see, halve the items must be transfered and since node capacities in SANs are restricted, such readjustments must possibly be done item wise or in small portions sequentially over all nodes. One can do slightly better by inserting the new node in the middle, but it does not change the fact that the fraction of items moved is independent of $|V|$. This example shows that if $\#I_v = 1$ and $s_{M_1} = 1$ is wanted it is inescapable that each subrange is involved in readjusts. So, if we forbid such global adjustments, we have to abandon an optimal smoothness and accept $s_{M_{1,\Lambda}} > 1$.

To achieve a $M_{c,\Lambda}$-decomposition we can roughly distinguished the method Λ into two categories. The first category is random based and introduced shortly in the following Section 3.1.1. The second category is a deterministic decomposition and introduced in Section 3.1.2, where our approach belongs to, see Section 3.2.

3.1.1 Random $M_{c,\Lambda}$-Decomposition

Since our focus is on deterministic decomposition, we give a brief introduction in random methods. The model for node insertion can be distinguished into simple single choice join and multiple choice join algorithms or games. As we have already seen in Section 2.1.3 these strategies are strongly related to DHTs. The single choice game is simply choosing the insertion point $r_v \in M$ uniformly at random, where each nodes owns one insertion point, we denote with $M_{1,1-random}$. One can show that after inserting n nodes in M we will have with high probability for each $I_v \in M_{1,1-random}$, $\Theta(\log n/n) \geq |I_v| \geq \Theta(\frac{1}{n \log n})$, which results to a smoothness of $s_{M_{1,random}} = \Theta(\log^2 n)$, see Lemma 1. The authors in Naor et. al [33] proposed an interesting multiple choice variant. Each node v observes $k = \mathcal{O}(\log n)$ positions in M. Then, v choses the point which has hit the largest range and places itself in the middle of this range. The resulting strategy of placing r_v in the middle, combined with k multiple choices we denote with $M_{1,k-randHalving}$. It leads with high probability to three different sized range classes, where $|I_v| \in \{2/n, 1/n, 1/2n\}$. According to this, the resulting smoothness is constant, in particular $s_{M_{1,k-randHalving}} = 4$ [26],[33].

From this we can draw the following conclusion. If we use randomness and do not want to control the smoothness by adjusting many node ranges, we have to enlarge the number of points a node owns or positions a node potentially can choose. Both will compensate the effect of badly chosen positions or too small or large ranges. Especially, for a small number of nodes the managing becomes inefficient and dominated by the number of copies. So, for a small number of nodes communication is not that expansive. Thus, searching for good positions by a number of choices or using broadcasts over all nodes is better than increasing the number of virtual nodes. However, if we increase the number of copies and combine this with multiple choices, we will obtain a lower smoothness. If we only increase the number of choices, we will end up in the deterministic greedy algorithm, we shortly introduce in the following Section 3.1.2. Finally, if the smoothness is high, we will need further balancing methods to compensate the imbalance. This is necessary if items are stored on node a with restricted capacities.

3.1.2 Deterministic $M_{c,\Lambda}$-Decomposition

The dilemma of dynamic deterministic decompositions is easy to comprehend if we require that $\#I_v = 1$ and forbid readjusts within the range. As long as we do not predefine $1 \leq n = |V|$ and allow variations in its cardinality we can not achieve for each n a good or optimal smoothness. So the idea is, if we make continuously slight errors, we can insert a node and coevally bound the smoothness, especially if we want to keep it constant. Further, we are interested in decompositions without dependancies between n and the fragmentation ratio Λ for new ranges. This is argued by the aim to arrange the decomposition of an appropriate interval with local information and additionally to keep the overhead for managing and finding an appropriate range small, thus $\#\{|I| \mid I \in M_c\}$ the number of different interval sizes should be constant. In the following we only consider M_1-decompositions, because if we obtain a good approximation with $c = 1$ it implies a straight forward solution for multiple intervals. There we would consider only $|I_v|$ and not the scattering of I_v, so sizes of responsibility ranges will only differentiate partially. In the following we denote a decomposition by $M_{c,\Lambda}$-dc, where Λ is an appropriate constant used in a deterministic method to generate the decomposition.

Subdividing with 1/2

A first naive deterministic $M_{1,\Lambda}$-dc can be obtained by using $\Lambda = 1/2$. A range for new node v is created by subdividing an exiting interval I_u into two ranges, where $|I_v| = \Lambda \cdot |I_u|$ and afterwards $|I_u| := |I_u| - |I_v|$. Therefore,, we assume that we have an algorithm $getMax(M_{c,\Lambda})$ and $getMin(M_{c,\Lambda})$ returning the largest or the smallest range with the least r_v and the appropriate node v. Further, each node knows its adjacency v_L and v_R. Note, from now on we will use this notation equivalent to links, as an example $v_{LR} = v = v_{RL}$, and v_{LL} is the left neighbor of v its left neighbor. In the following we consider item movements only if they are caused by dynamical changes within the decomposition and we do not reassign any r_d or change the distribution of items in M.

1. $insertNode(M_{1,\frac{1}{2}}, v)$
   determine $I_u = getMax(M_{1,\frac{1}{2}})$, then set $r_v := (I_u.start + I_u.end) \cdot \frac{1}{2}$ and set $I_v := [r_v, I_u.end[$ and $I_u := [r_u, r_v[$. Move all items $d \in f^{-1}(u)$ to v, where now $r_d \notin I_u$.

2. $deleteNode(M_{1,\frac{1}{2}}, v)$
   So if $|I_v| = 2^{-\lceil \log_2 n \rceil}$ and $(r_d/|I_v|) \mod 2 = 0$, move all $d \in f^{-1}(v)$ to v_R

and set $I_{v_R} := [r_v, I_{v_R}.end]$, thus we set $r_{v_R} = r_v$, or if $(r_d/|I_v|) \mod 2 = 1$ move all $d \in f^{-1}(v)$ to v_L and set $I_{v_L} := [r_{v_L}, r_{v_R}[$.
Otherwise, if $|I_v| = 2^{-\lfloor \log_2 n \rfloor}$ and $\log_2 n \notin \mathbb{N}$ then delete $u := getMin(M_{1,\frac{1}{2}})$ set $r_u := r_v$ and $I_u := I_v$ and move all $d \in f^{-1}(v)$ to u.

These algorithms do not consider runtime, but they exemplify roughly the idea. As long as we subdivide only one of the largest ranges we know $1 \leq \#\{|I| \mid I \in M_{1,\frac{1}{2}}\} \leq 2$ and $|I_v| \in \{2^{-\lfloor \log_2(n-1) \rfloor}, 2^{-\lceil \log_2(n-1) \rceil}\}$. So we sustainedly have a smoothness of $s_{M_{1,\frac{1}{2}}} = 2^{-\lceil \log_2(n-1) \rceil}/2^{-\lfloor \log_2(n-1) \rfloor} = 2$, except if $\log_2 n \in \mathbb{N}$.

Corollary 13. *The $M_{1,\Lambda}$-decompositions with $\Lambda = 1/2$ has a worst case smoothness of 2 and the decomposition is optimal if $n = 2^{\lfloor \log_2 n \rfloor}$. Node insertion is monotone, but deletion in some cases not.*

Basically, we can keep a smoothness of 2 quit easily. Therefore, we have to be careful in case of *deleteNode*(). For preserving smoothness, if $|I_v| = 2^{-\lfloor \log_2 n \rfloor}$, it requires to chose an appropriate substitute. This substitution causes the none monotone behavior. More freedom in choosing nodes for insertion or deletion substitution will increase the managing overhead and will require a more complex algorithms. In the following section we consider a deterministic decomposition, where the managing effort is reduced by using a function for partitioning and position choice.

Decomposing with Linear Hash Functions

The motivation of this approach is founded by a commonly known hash method called Fibonacci Hashing. The Fibonacci Sequence $fib(n)$ is a very popular and well studied sequence and defined as follows

$$\begin{aligned} fib(0) &:= 0 \\ fib(1) &:= 1 \\ fib(n) &:= f(n-1) + f(n-2) \end{aligned}$$

An interesting attribute of the Fibonacci Number is that the ratio of two consecutive Fibonacci Numbers,

$$\lim_{n \to \infty} \frac{fib(n+1)}{fib(n)} = \frac{1+\sqrt{5}}{2}$$

tend to an irrational number, which is also known as the Golden Number $\Phi := 2^{-1}(1+\sqrt{5}) \approx 1.618$. Remember, we want to obtain a decomposition that subdivides ranges asymmetrically, but with a constant number of distinct interval

sizes. This we can describe by solving the following equation

$$c^{-i} = c^{-(i+1)} + c^{-(i+2)} \Leftrightarrow c^2 = c + 1$$

not surprisingly solved by $c \in \{\Phi, (1-\sqrt{5})2^{-1}\}$. From this we can derive the equation

$$1/\Phi^i = 1/\Phi^{i+1} + 1/\Phi^{i+2} \tag{3.1}$$

This implies to split ranges recursively such that $|I_v| = |I'_v| + |I_u|$ and $|I'_v|/|I_u| \in \{\Phi, \Phi^{-1}\}$. Finally, if we guarantee to select and partition only $getMax(M_{1,\Phi})$, we know $1 \leq \#\{|I| \mid I \in M_c\} \leq 3$. Further, this decomposition has a smoothness $s_{M_{1,\Phi}} = \Phi^2 \approx 2.618$.

This smoothness is worse than before in Section 3.1.2, but we can decrease managing efforts by applying Linear Hash Functions. Therefore, we set $f(v) := h(v.key) = (v.key \cdot \Phi^{-1}) \mod 1$ and nodes are identified by integers in the range of $v.key \in \{0, n-1\}$ and we require $\forall v \neq u \; v.key \neq u.key$. A specific attitude of this hashing is, that it always hits one of the biggest remaining intervals [22]. This also implies only at most 3 different classes of interval sizes. So, a smoothness of $s_{M_{1,f}} \in \{\Phi^{-i}/\Phi^{-(i+1)}, \Phi^{-i}/\Phi^{-(i+2)}\}$ is guaranteed if we keep the key space compact. In contrast to the algorithms in Section 3.1.2, inserting and deleting is quite simple, because all we have to do is to keep the used key space compact and use the hash function to determine r_v. In case of deleting, we simply use the node with the highest key as substitute. This leads to the following algorithms describing how to handle dynamics by coevally keeping the smoothness of $M_{1,f}$. As before, we consider item movements only if they are caused by dynamical changes within the decomposition and we do not reassign any r_d or change the distribution of items in M.

1. $insertNode(M_{1,f}, v)$
   Determine $n = |V| = |M_{1,f}|$ and compute $r_v := (n+1) \cdot \Phi^{-1} \mod 1$. Set links to $v_L := u$ where $r_u = max_{u \in V}(\{r_u\}) < r_v$ and $v_R := u$ where $r_u = min_{u\ inV}(\{r_u\}) > r_v$ and $I_v := [r_v, r_{v_R}[$ and $I_{v_L} := [r_{v_L}, r_v[$. Move all $d \in f^{-1}(v_L)$ to v, if $r_d \in I_v$.

2. $deleteNode(M_{1,f}, v)$
   determine $r_u := n \cdot \Phi^{-1} \mod 1$ and move all $d \in f^{-1}(u)$ to u_L and set $I_{u_L} := [r_{u_L}, r_{u_R}[$ and move all $d \in f^{-1}(v)$ to u. Set $u.key := v.key$ and $r_u := u.key \cdot \Phi^{-1} \mod 1$. Set $u_L := v_L$ and $u_R := v_R$, thus $I_u := [r_u, r_{v_R}[$. Move all $d \in f^{-1}(v)$ to u.

To determine the nodes, we assume that we can access an additional search structure sorted over r_v.

Corollary 14. *The $M_{1,\Lambda}$-decompositions with $\Lambda = f(v)$ for $n > 1$ has a worst case smoothness of $\Phi^2 \approx 2.618$ and a best case smoothness of $\Phi \approx 1.618$. Node insertion is monotone, but deletion not.*

We can observe that linear hashing using Φ ensures that each new injection point hits one of the largest remaining intervals. Therefore, we have to keep the key space compact. Unfortunately, this results in unthrifty delete operations.

3.2 Deterministic Greedy Decomposition

In contrast to random sampling and subdividing with hash functions, we introduce in this section deterministic and greedy decompositions, based on the the Golden Section. Similar to Section 3.1.2 our aim is to generate only a small and constant number of different intervals. Further, we want to improve the smoothness of the simple $M_{1,1/2}$-decomposition, so we try to obtain $s_{M_{c,\Lambda}} < 2$.

3.2.1 The Φ-*Div* Algorithm

The following approach is a transformation of linear hashing of Section 3.1.2 to a greedy algorithm using Φ. We still require $\#I_v = 1$ for the $M_{1,\Phi\text{-}Div}$-decomposition, so the algorithm runs without copies or any other kind of rebalancing. Φ-*Div* has slightly more freedom in choosing intervals for inserting nodes and finding substitutes for deleting nodes. This freedom of choice results into less managing overhead, because we do not have to care about the compactness of a key space as linear hashing needs to.

We assume for Φ-*Div* listed in Algorithm 6 that each node v has a link to the adjacent nodes v_L, v_R in M, owning $I_{v_L}, I_{v_R} \in M_{1,\Phi\text{-}Div}$. The main part of the Φ-*Div* is to organize all intervals in $M_{1,\Phi\text{-}Greedy}$ and to select and partition them via Φ. To manage node insertions Φ-*Div* requires three sets, where each set

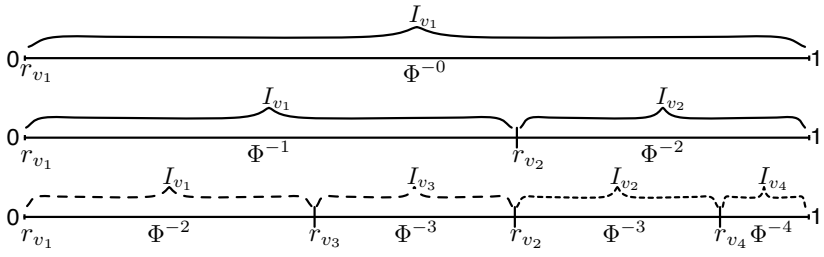


Figure 3.1: Decomposition Example of M with Φ.

stores references to nodes owning same sized intervals. In particular L_Φ^1 refers to nodes of size Φ^{-i}, L_Φ^2 of size $\Phi^{-(i+1)}$, and L_Φ^3 of size $\Phi^{-(i+2)}$. These three sets are sufficient follows directly from the Φ-*Div* algorithm and the subsequent Lemma 7. These sets must be implemented in a distributed or central data structure, which depends on the application area. We do not consider the operations or costs to maintain these sets and the membership of nodes. The resulting Φ-*Div* implements a single node insertion and the necessary managing operations for items. A decomposition example of M generated by Φ-*Div* having fixed node positions and a constant number of different interval is pictured in Figure 3.1.

Lemma 7. *If $|V| \geq 2$, then Φ-Div needs to handle at most three different types of interval lengths in $M_{1,\Phi\text{-}Div}$, so*

$$2 \leq |\{|I_v|, I_v \in M_{1,\Phi\text{-}Div}\}| \leq 3$$

Proof. Assuming the greatest interval in M is of length $|I_{max}|$, before node v is inserted. To insert v, Φ-*Div*takes any interval from $I_u \in L_\Phi^1$, containing intervals with $|I_u| = |I_{max}|$ and creates two intervals of length $1/\Phi^{i+1}$ and $1/\Phi^{i+2}$. This is guaranteed by attributes of Φ, because $1/\Phi^i = 1/\Phi^{i+1} + 1/\Phi^{i+2}$. If after inserting v an interval of length $1/\Phi^{i+3}$ is produced the algorithm must have chosen wrong and the size of $|I_{max}|$ was $1/\Phi^{i+1}$, which contradicts the algorithm and the

Algorithm 6 Φ-*Div*: Inserting a new node v into M

Require: Φ, L_Φ^1, L_Φ^2, L_Φ^3, v
1: **if** $(\sum_{i=1}^{3} L_\Phi^i.nodeCount() == 0)$ **then**
2: $r_v := 0; v_L := v; v_R := v;$ /* $\Leftrightarrow |I_v| = 1$*/
3: $L_\Phi^1.add(v);$
4: **else**
5: $u := L_\Phi^1.removeAnyNode();$
6: $v_L := u; v_R := u_R; u_R := v; \delta := (r_{v_R} - r_u)/\Phi;$
7: **chose** /*remember one ratio, but two solutions*/
8: a) $r_v := r_u + \delta$
9: b) $r_v := r_{v_R} - \delta;$
10: **if** (a) **then** $L_\Phi^2.add(u); L_\Phi^3.add(v);$
11: **else** $L_\Phi^2.add(v); L_\Phi^3.add(u);$
12: move all $d \in f^{-1}(u) \mid r_d \in [r_v, r_{v_R}[$ to $v;$
13: **end if**
14: **if** $L_\Phi^1.isEmpty$ **then**
15: $L_\Phi^1 = L_\Phi^2; L_\Phi^2 = L_\Phi^3; L_\Phi^3 = L_\Phi^1;$ /* fixing the sets if L_Φ^1 runs empty*/
16: **end if**

assumption. If L^1_Φ runs empty after an insertion, the set indices are redefined. Thus, L^1_Φ and L^2_Φ can never run empty. □

Theorem 14. *The $M_{1,\Phi\text{-}Div}$-decomposition generated by Φ-Div has a smoothness of $1.618 \approx \Phi \leq s_{M_{1,\Phi\text{-}Div}} \leq \Phi^2 = 1 + \Phi \approx 2.619$ for any $n > 1$ and the number of steps to insert a node is constant.*

Proof. By Lemma 7 it follows that only intervals of size $1/\Phi^i$, $1/\Phi^{i+1}$, and $1/\Phi^{i+2}$ can appear. The upper smoothness bound of Φ^2 occurs, as long as L^1_Φ and L^3_Φ are not empty. The lower smoothness bound appears, if the last node v with $|I_v| = |I_{max}|$ in L^1_Φ is subdivided. So only intervals of length $1/\Phi^{i+1}$ and $1/\Phi^{i+2}$ are remaining. Further, because of Lemma 7 it is sufficient to take any node from L^1_Φ, which can be done with the remaining steps in $\mathcal{O}(1)$. □

The main achievement of this algorithm is that we do not have to organize a key space or search for an appropriate insertion point, and we do not have to find for each delete operation a substitute. However, if Φ-*Div* should be used in a distributed fashion, we need an efficient membership implementation for the sets, because a modified Φ-*Div* will have to use broadcasts for keeping nodes up to date if sets are relabeled.

Since we want to cope dynamics we also have to name an algorithm to remove nodes. Node deletions and preserving Φ-*Div* runtime conditions will require to repair the decomposition afterwards. Therefore, we have to remap the unassigned I_v to a substitute u and join I_v and I_v to a single interval. In some cases this must be done carefully. We cannot always remove v by simply selecting a neighbor as a substitute. This will violate the smoothness bounds if joined intervals become to large. Further, during the decomposition it might happen that v has an adjacency with $|I_v| = |I_{v_L}| = |I_{v_R}|$. Such a neighbor selection will produce a joined interval of size $2 \cdot \Phi^{-i} \neq \Phi^{-(i-1)}$, according to Lemma 7 and Equation 3.1 such a join will cause an unwanted change of Φ-*Div* preconditions. First, we define consecutive sequences of intervals with equal sizes generated by any Λ. For simplification we consider it within a range M and not within a ring M.

Definition 24. *For any $M_{c,\Lambda}$ decomposition we denote with S_I the set of consecutive intervals $I \in M_{c,\Lambda}$ of size $|I|$ and its cardinality with $\#S_I$. So*

$$\sum_{I \in S_I} |[I.start, I.end[| = |[min(\{I.start\} \mid I \in S_I), max(\{I.end\} \mid I \in S_I)[|$$

To bound and solve the problem caused by joining intervals of leaving nodes and their substitutes, we have to take a look at the S_I and especially $\#S_I$ for each $I \in M_{1,\Phi\text{-}Div}$. This is described by the following lemma.

Lemma 8. *For any sequence S_I, with $I \in M_{1,\Phi\text{-}Div}$, where $|I| \in \{\Phi^{-i}, \Phi^{-(i+1)}, \Phi^{-(i+2)}\}$ the following holds: If $I \in L^1_\Phi \vee I \in L^2_\Phi$ then $\#S_I \leq 4$ and if $I \in L^3_\Phi$ then $\#S_I \leq 2$.*

Proof. First if $I \in L^1_\Phi \vee I \in L^2_\Phi$: Assuming we have 2 adjacent intervals $I, I' \in M$ with $|I| = |I'| = 1/\Phi^i$. Based on the freedom given in line 8 and 9 of Algorithm 6 and Equation 3.1 a further insertion in I by Φ-*Div* of 2 additional nodes implies 2 intervals of size $|1/\Phi^{i+2}|$ and 1 of size $|1/\Phi^{i+3}|$ in I with 3 possible final interval arrangements, and 1, where both bigger intervals are nearer to I'. This can be done analogously, but mirrored for I' which results in 4 successive intervals of size $|1/\Phi^{i+2}|$ surrounded by 2 intervals of size $|1/\Phi^{i+3}|$. This can be used as recursive substitution for any interval pair of equal size.

Second, if $I \in L^3_\Phi$: Assume both adjacent intervals I_{v_L}, I_{v_R} are members of L^3_Φ. Based on the first part and Equation 3.1 we know that either I_{v_L} or I_{v_R} has an adjacent interval of size $|I|^{-(i+3)}$ which contradicts the assumption $I \in L^3_\Phi$. So each $I \in L^3_\Phi$ has an adjacent interval of size $\Phi^{-(i+1)}$. □

Note, Lemma 8 only says that by discovering the neighborhood we will find within a constant number of steps a different sized interval. The Lemma does not say anything about the number of steps needed to find an interval of specific size.

The algorithm to accomplish a node deletion operation is denoted by Φ-*Del* and the main steps and cases to distinguish are listed in the following. We regard these operations based on the set membership of a node according to Φ-*Div*. The description makes use of the adjacencies bounds of Lemma 8 for removals and Equation 3.1. The principles of item reassignment afterwards are the same if we intent to keep $\#I_v = 1$, likewise as applied in Φ-*Div* and other algorithms of this chapter.

1. Φ-*Del*(v) *if* $(v \in L^3_\Phi)$:

   Chose a $u \in L^2_\Phi$ adjacent to v and move all $d \in f^{-1}(v)$ to u. Join I_v and I_u as new I_u, remove u from L^2_Φ and add it to L^1_Φ. Further, remove v from L^3_Φ and set the adjacency links of u and finally, if $u = v_R$ then $r_u := r_v$.

2. Φ-*Del*(v) *if* $(v \in L^2_\Phi)$:

   If $L^3_\Phi \neq \emptyset$ and $\{v_L, v_R\} \cap L^3_\Phi \neq \emptyset$, then chose $u \in \{v_L, v_R\} \cap L^3_\Phi$. Remove u from L^3_Φ and v from L^2_Φ. Join I_v and I_u as new I_u and insert u in L^1_Φ. Finally, move all $d \in f^{-1}(v)$ to u, set the adjacency links, if $u = v_R$ then $r_u := r_v$.

   But if $L^3_\Phi \neq \emptyset$ and $v_L, v_R \notin L^3_\Phi$, then chose any $u \in L^3_\Phi$ and remove it with Φ-*Del*(u). Insert u as substitute at r_v by moving all $d \in f^{-1}(v)$ to u. Finally, set the adjacency links and the new r_u.

Else if $L^3_\Phi = \emptyset$ and $\{v_L, v_R\} \cap L^1_\Phi \neq \emptyset$, then chose $u \in \{v_L, v_R\} \cap L^1_\Phi$. Remove u from L^1_Φ and v from L^2_Φ, join I_v and I_u as new I_u. Relabel L^2_Φ to L^3_Φ and L^1_Φ to L^2_Φ and establish a new L^1_Φ by inserting I_u into it. Finally, move all $d \in f^{-1}(v)$ to u and set the adjacency links, if $u = v_R$ then $r_u := r_v$.

But if $L^3_\Phi = \emptyset$ and $v_L, v_R \notin L^1_\Phi$, then discover the neighborhood links for a $u \in L^2_\Phi$, where $\{u_L, u_R\} \cap L^1_\Phi \neq \emptyset$. Remove u with Φ-$Del(u)$ and insert u as substitute at r_v by moving all $d \in f^{-1}(v)$ to u. Finally, set the adjacency links and the new r_u.

3. Φ-$Del(v)$ *if* $(v \in L^1_\Phi)$:

   If $L^3_\Phi \neq \emptyset$, remove any node $u \in L^3_\Phi$ with Φ-$Del(u)$ and insert u as substitute at r_v by moving all $d \in f^{-1}(v)$ to u. Finally, set the adjacency links and the new r_u.

   All other cases are similar as before. We have to find an appropriate substitute in the neighborhood or any $u \in L^2_\Phi$. Remove it and insert it at r_v with new adjacency links. During this items must be transferred and we have relabeled the sets, because the removal of u has produced a new interval class of size $|I_u|$.

Note, if the algorithm is used in a distributed environment the search for appropriate substitutes can be supported by discovering the neighborhood simultaneous among the links v_L and v_R. This is reasonable, because Lemma 8 implies that in some cases we might find a node that fits within a constant number of hops. Eventually Φ-*Div* and Φ-*Del* allow us to generate a dynamical and nearly balanced decomposition for item allocations in uniform environments. Independent of the even item distribution, the current smoothness bound still requires to apply further techniques to compensate this deviation. This can be done by increasing $\#I_v$ or rebalance an item distribution with multiple choice by using runtime information, more on using balancing in Chapter 4.

3.2.2 The Φ-Div^+ Algorithm

As seen in Theorem 14, Φ-*Div* is no real opponent compared to the worst case bounds of the naive symmetric approach from Section 3.1.2. In this section our major aim is to give an improved deterministic greedy algorithm, which achieves better smoothness bounds in worst and in best case.

The problem of the previous algorithm is that we can not avoid the existence of three different interval sizes if we use Φ. However, to improve the smoothness we have to deal with it in a different way as we have done before. The idea is to

loose the ties and allow some local adjusts in the neighborhood, by changing the interpretation of intervals. The goal of these adjustment is to decrease the size of responsibility areas of all nodes owning a range in L^1_Φ and to increase them in L^3_Φ. Nevertheless, we do not want to abandon the simple interval management. To reach this we have to overcome the direct coupling of node injections and intervals receiving items. Therefore, we establish a further virtual decomposition layer $M'_{1,\Lambda}$ on top of $M_{1,\Lambda}$-dc. This we do by placing an anchor for each node within its $I_v \in M_{1,\Lambda}$. This will allow us to do some local adjusts within $M'_{c,\Lambda}$ without changing $M_{1,\Lambda}$. The anchor we use is defined in the following.

Definition 25. *For each $I_v \in M_{1,\Lambda}$, we denote with $a_v \in I_v$ the anchor of a node v in the range I_v. Its position is determined by $r_v + |I_v| \cdot a$, where $a \in [0, 1[$.*

Note, the anchor also identifies an additional logical position for each node in M. This definition is extendable for $\#I_v > 1$, but because of simplification and our scenario not needed. So a_v is a virtual and floatable position narrowed by the range I_v. Based on a_v we can redefine the item responsibility range of a node. Without loss of generality we give the definition for a hash range and not within a unit ring. The adjacency of v is still v_L and v_R.

Definition 26. *Let $M_{1,\Lambda}$-dc a decomposition of M and v the node owning $I_v \in M_{1,\Lambda}$, then the range $I'_v := [(a_v + a_{v_L})/2, (a_v + a_{v_R})/2[$ defines the responsibility range of v and $M'_{1,\Lambda}$-dc denotes the decomposition based on I'_v.*

With Definition 25 and 26 we can introduce an additional dynamic decomposition, using a virtual segmentation for an item assignment. To achieve a decomposition we inherit the segmentation factor from Φ-Div, but we reduce the freedom in choosing r_v. Compared to Algorithm 6 in line 8 and 9 we only use one solution of Equation 3.1, the benefit of this we will see later in Lemma 8. From now on we chose a segmentation in such way, that the bigger remainder after inserting u is always left. Note, the following results will also hold if we chose 'always right'.

Both together, the additional layer and the restricted segmentation rules of intervals leads to the algorithm Φ-Div^+ and the resulting decomposition $M_{1,\Phi\text{-}Div^+}$. Finally, an item $d \in D$ is mapped into the hash range at r_d and stored on v if $r_d \in I'_v$. This implies if a node is inserted into the decomposition or removed from the decomposition, the reassignments are decided by I'_v. So it remains to determine an appropriate a for the anchor position. In our scenario we set $a := 1/2$ for all nodes independent of $|I_v|$, $|I_{v_R}|$, or $|I_{v_L}|$. This allows us to operate locally and to border the readjustments effects in the neighborhood. The algorithm Φ-Div^+ for inserting a single node is listed in Algorithm 7.

Algorithm 7 Φ-Div^+ : Inserting v into M

Require: $a := 1/2$, Φ, L^1_Φ, L^2_Φ, L^3_Φ, v
1: **if** $(\bigcup_{i=1}^{3} L^i_\Phi == \emptyset)$ **then**
2: $r_v := 0; v_L := v; v_R := v;$ // $\Leftrightarrow |I_v| = 1$
3: $L^1_\Phi.add(v);$
4: **else**
5: $u := L^1_\Phi.removeAnyNode();$
6: $v_L := u; v_R := u_R; u_R := v; \delta := (r_{v_R} - r_u)/\Phi;$
7: $r_v := r_u + \delta;$ // bigger is always left; v obtains the smaller side
8: $a_v := (r_v + r_{v_R})/2;$ // because a==1/2
9: $I'_v := [(a_v + a_{v_L})/2, (a_v + a_{v_R})/2[$ // also adapt I'_{v_L} and I'_{v_R}
10: $L^2_\Phi.add(u); L^3_\Phi.add(v);$
11: move all $d \in f^{-1}(v_L) \mid r_d \in I'_v$ to v;
12: move all $d \in f^{-1}(v_R) \mid r_d \in I'_v$ to v;
13: **end if**
14: **if** $L^1_\Phi.isEmpty$ **then**
15: $L^1_\Phi = L^2_\Phi; L^2_\Phi = L^3_\Phi; L^3_\Phi = \emptyset;$ // fixing the sets if L^1_Φ runs empty
16: **end if**

In the following we take a look at the properties of the $M_{1,\Phi\text{-}Div^+}$-dc. We will analyze the smoothness of the virtual layer, now responsible for item assignments. Therefore, we first answer if the algorithm still produces only three different interval types in the decomposition.

Lemma 9. *If $|V| \geq 2$, then Φ-Div^+ needs to handle at most 3 different types of interval lengths in $M_{1,\Phi\text{-}Div^+}$, so*

$$2 \leq |\{|I_v|, I_v \in M_{1,\Phi\text{-}Div^+}\}| \leq 3$$

Proof. This follows directly from Lemma 7 and the further facts. First, the additional layer has no impact on the decomposition. Second, Lemma 7 holds as long as one solution of Equation 3.1 is used. So it is valid to use always the same. □

Second, we have to check which impact the restricted injection point choice, compare Algorithm 6 Line 8, 9 with Algorithm 7 Line 7, has on $\#S_I$, $I \in M_{1,\Phi\text{-}Div^+}$. This is described in the following lemma.

Lemma 10. *For any sequence S_{I_v}, with $I_v \in M_{1,\Phi\text{-}Div^+}$ the following holds: $\#S_{I_v} \leq 2$ and only if $|I_v| = min(\{|I|, I \in M_{1,\Phi\text{-}Div^+}\})$ then $\#S_{I_v} = 1$.*

Proof. Assuming we have 2 adjacent interval $I_v, I_u \in M_{1,\Phi\text{-}Div^+}$ with $v_R = u$ and $|I_v| = |I_u| = \Phi^{-i}$. According to Φ-Div^+ Line 7 $|I_{v_L}|$ is either of size $\Phi^{-(i-1)}$ or $\Phi^{-(i+1)}$. This is, because it is the smaller part of a previous segmentation of size $\Phi^{-(i-2)}$, otherwise Φ-Div^+ has chosen the wrong solution of Equation 3.1. From this we know that $|I_{u_R}| = \Phi^{-(i+1)}$.

This can also be reconstructed by building a binary tree, where the root has the number $i \in \mathbb{N}$ and the left child node the number $i+1$ and the right $i+2$. According to Φ-Div^+ , where the number is the negative power of Φ, we add child leafs only where the leaf has one of the smallest leaf numbers. Clearly only three distinct numbers can occur within the leafs. If we print only the leaf numbers during a depth search, the traversal output has the same property. Two equal leaf numbers can only occur if both have different direct parent nodes. This excludes sequences longer than 2.

A similar argument holds for the leaf v with the highest number $i+2$. It always has an adjacent leaf with the number $i+1$, if they have the same parent with the number i. According to Φ-Div^+ it must occur at the left side. If not, and the number on the left is equal, then both can not have the same direct parent. Further, according to Φ-Div^+ the right leaf must have the number $i+3$, which contradicts the assumption that v has the highest number. □

With Lemma 9 and Lemma 10 we can proof the following theorem. It mainly considers the smoothness of the additional layer, now responsible for an item assignment. It consists of intervals I'_v from Definition 26 produced by Φ-Div^+ , where the anchor of v is placed at $a_v := r_v + |I_v| \cdot 1/2$.

Theorem 15. *For $|V| > 2$ each $I \in M_{1,\Phi\text{-}Div^+}$, $\#S_I \leq 2$ and $2 \leq \#\{|I|\} \leq 3$. For $M'_{1,\Phi\text{-}Div^+}$-dc the smoothness is bounded by*

$$1.118 \approx \frac{1}{\Phi} + \frac{1}{2} \leq s_{M'_{1,\Phi\text{-}Div^+}} \leq \frac{1+\Phi^2}{2} \approx 1.809.$$

Proof. First, the number of distinct intervals follows from Lemma 9 and the sequence statement from Lemma 10.

Note, in the following we will use the facts that $1 + \Phi^{-1} = \Phi$, $1 + \Phi = \Phi^2$, and Equation 3.1. Further we will use the binary tree representation of Lemma 10. The resulting cases can be retraced in Figure 3.2.

Second, the lower bound: It occurs if $|V|$ is a Fibonacci Number. Then only two distinct numbers can be found in the binary tree leafs. One can prove this by induction, if we count the occurrence of each number in the leafs if $|V|$ is a Fibonacci Number. The resulting count for both is also a Fibonacci Number. So the largest interval $I'_v \in M'_{1,\Phi\text{-}Div^+}$ is built upon subsequent intervals $I_{v_L}, I_v, I_{v_R} \in$

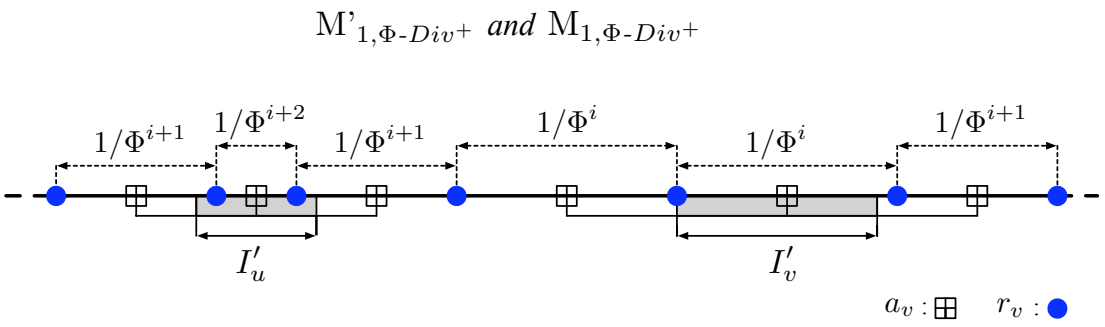


Figure 3.2: Smoothness Bound Construction

$M_{1,\Phi\text{-}Div^+}$, where $|I_{v_L}| = \Phi^{-(i+1)}$, $|I_v| = \Phi^{-i}$, and $|I_{v_R}|\Phi^{-i}$ (or $|I_{v_L}| = \Phi^{-i}$, $|I_v| = \Phi^{-i}$, and $|I_{v_R}|\Phi^{-(i+1)}$). This follows from Lemma 10 and leads to a size of

$$\begin{aligned} |I'_{max}| &= max(\{|I'_v|, I'_v \in M'_{1,\Phi\text{-}Div^+}\}) \\ &= \frac{1}{2}\frac{1}{\Phi^i} + \frac{1}{2}\left(\frac{1}{2\Phi^{i+1}} + \frac{1}{2\Phi^i}\right) \\ &= \frac{1}{4\Phi^i}\left(2 + 1 + \frac{1}{\Phi}\right) \\ &= \frac{1}{4\Phi^i}\left(1 + \Phi^2\right) \end{aligned}$$

Further, the smallest interval $I'_u \in M'_{1,\Phi\text{-}Div^+}$ is constructed by subsequent intervals $I_{u_L}, I_u, I_{u_R} \in M_{1,\Phi\text{-}Div^+}$, where $|I_{u_L}| = \Phi^{-i}$, $|I_u| = \Phi^{-(i+1)}$, and $|I_{u_R}|\Phi^{-i}$. This also follows from Lemma 10 and leads to

$$\begin{aligned} |I'_{min}| &= min(\{|I'_u|, I'_u \in M'_{1,\Phi\text{-}Div^+}\}) \\ &= \frac{1}{2}\left(\frac{1}{2\Phi^i} + \frac{1}{2\Phi^{i+1}}\right) + \frac{1}{2}\left(\frac{1}{2\Phi^{i+1}} + \frac{1}{2\Phi^i}\right) \\ &= \frac{1}{2\Phi^i}\left(2 + 1 + \frac{1}{\Phi}\right) \\ &= \frac{1}{4\Phi^i}2\Phi \end{aligned}$$

From this we obtain

$$\begin{aligned}\frac{max(\{|I'_v|, I'_v \in M'_{1,\Phi\text{-}Div^+}\})}{min(\{|I'_u|, I'_u \in M'_{1,\Phi\text{-}Div^+}\})} &= \frac{1+\Phi^2}{4\Phi^i} \Big/ \frac{2\Phi}{4\Phi^i} \\ &= \frac{1}{\Phi} + \frac{1}{2} \\ &\approx 1.118\end{aligned}$$

Third, the upper bound: It occurs if three distinct numbers can be found in the leafs. In this case the largest interval $I'_v \in M'_{1,\Phi\text{-}Div^+}$ is the same we have constructed for the lower case proof. So we have

$$\begin{aligned}|I'_{max}| &= max(\{|I'_v|, I'_v \in M'_{1,\Phi\text{-}Div^+}\}) \\ &= \frac{1}{4\Phi^i}\left(1+\Phi^2\right)\end{aligned}$$

Similar as before the smallest interval $I'_u \in M'_{1,\Phi\text{-}Div^+}$ is constructed by subsequent intervals $I_{u_L}, I_u, I_{u_R} \in M_{1,\Phi\text{-}Div^+}$, but now $|I_{u_L}| = \Phi^{-(i+1)}$, $|I_u| = \Phi^{-(i+2)}$, and $|I_{u_R}| = \Phi^{-(i+1)}$. This also follows from Lemma 10 and leads to

$$\begin{aligned}|I'_{min}| &= min(\{|I'_u|, I'_u \in M'_{1,\Phi\text{-}Div^+}\}) \\ &= \frac{1}{2}\left(\frac{1}{2\Phi^{i+1}} + \frac{1}{2\Phi^{i+2}}\right) + \frac{1}{2}\left(\frac{1}{2\Phi^{i+2}} + \frac{1}{2\Phi^{i+1}}\right) \\ &= \frac{1}{4\Phi^i} 2\left(\frac{1}{\Phi} + \frac{1}{\Phi^2}\right) \\ &= \frac{2}{4\Phi^i}\end{aligned}$$

From this we obtain

$$\begin{aligned}\frac{max(\{|I'_v|, I'_v \in M'_{1,\Phi\text{-}Div^+}\})}{min(\{|I'_u|, I'_u \in M'_{1,\Phi\text{-}Div^+}\})} &= \frac{1+\Phi^2}{4\Phi^i} \Big/ \frac{2}{4\Phi^i} \\ &= \frac{1+\Phi^2}{2} \\ &\approx 1.809\end{aligned}$$

□

In the beginning of this section we have asserted that we can handle dynamics too. So we have to think about the removal of nodes. Actually, we can state that $\Phi\text{-}Div^+$ generating the decomposition $M_{1,\Phi\text{-}Div^+}$ is one possible outcome

of $M_{1,\Phi\text{-}Div}$. This is, because we only have used Φ-Div in such way, that we always chose Line 8 of Algorithm 6. According to this, Φ-Div^+ also produces at most 3 different types of intervals. So for removing nodes from $M_{1,\Phi\text{-}Div^+}$, Φ-Del principally would work, but needs some adaptions. Clearly, the item reassignment now is decided by intervals in $M'_{1,\Phi\text{-}Div^+}$ and not as before by intervals in $M_{1,\Phi\text{-}Div}$. Further, the constraint in choosing r_v implies a constraint for deleting r_v. We have to find in some more cases an appropriate substitute which does not violate the smoothness bounds of Theorem 15 and the precondition of Φ-Div^+ . This is, because due to the insertion constraint we can not join intervals where $|I_{v_L}| \leq |I_v|$. One can see this by constructing the according binary tree used in Lemma 10. Interval joins of two adjacent nodes are only possible if the according leafs in the tree share the same direct parent node. With respect to this the removal of nodes will work under the same conditions as it does in Φ-Del.

Eventually, the main achievements is the compensation of imbalance caused by leaving or joining nodes. The adjustments and their caused imbalance gets always compensated by two nodes. Of further importance is, that the amount of items a node receives or gets rid of is automatically weighted by the size of the adjacent nodes. Which finally results in the advanced smoothness.

Two Final Notes On The Additional Virtual Layer

So why do we have to built such a construction using an anchor? And why do we halve the distance between the anchors of nodes?

The first question we can answer by regarding an alternative decomposition without using the additional anchors, commonly found in DHTs. Here the nodes define their responsibility ranges by minimizing the distance between two injection points. Actually, this is simply halving the distances between adjacent node positions in the hash range M. So a nodes' range in DHT is defined by $I_v^* := [(r_{v_L} + r_v)/2, (r_{v_R} + r_v)/2[$. From Lemma 10 we know that one of the longest subsequent range sequence is like $|I_{v_L}| = \Phi^{-(i+1)}$, $|I_v| = \Phi^{-i}$, and $|I_{v_R}| = \Phi^{-i}$. By using the range definition of I_v^* the node v_R will receive one of the biggest range size of $|I^*_{max}| := |I^*_{v_R}| = \Phi^{-i}$. Further, one of the smallest range is obtained by a range sequence like $|I_{v_L}| = \Phi^{-(i+2)}$, $|I_v| = \Phi^{-(i+1)}$, and $|I_{v_R}| = \Phi^{-(i+2)}$. Applying the range definition I_v^* again we obtain $|I^*_{min}| := |I_v^*| = \Phi^{-i}/2 \cdot (\Phi^{-2} + \Phi^{-1}) = \Phi^{-i}/2$. Finally, this results into a worst case of $|I^*_{max}|/|I^*_{min}| = 2$. So, by using the anchor we are able to bridge the gab of two subsequent ranges, and it guarantees that each v with $|I_v| = \Phi^{-i}$ gets reduced, which finally results into a smoothness below 2.

To answer the second question we first have to remember that we do not want to care about the current range sizes in an adjacency. Each node should always

do the same operation to decide its responsibility for items. By choosing a different ratio than $1/2$ we have to establish a general partitioning rule like *the smaller takes from the bigger one*. Further, if we do not want to endanger the smoothness bounds during dynamics, we must ensure that the smallest and the largest sequence do not produce ranges with worst ratio. To ensure this for all nodes we have to consider a $v \in L^2_\Phi$ surrounded by nodes in $v_L, v_R \in L^3_\Phi$. Due to symmetry we only need to look at one side leading to the following equation (for better comprehension not simplified), which must hold for any c

$$\frac{1}{2}\left(\Phi^{-(i+1)} - \frac{\Phi^{-(i+1)} + \Phi^{-(i+2)}}{c}\right) \geq \frac{1}{2}\left(\Phi^{-(i+2)} + \frac{\Phi^{-(i+1)} + \Phi^{-(i+2)}}{c}\right)$$

Since our goal is to maximize $|I'_w|$ we have chosen $c = 2$, which leads to equality. If we chose a $c > 2$ then I'_v defines I'_{min} and increases the smoothness in worst case. Further, if $|V|$ is a Fibonachi Number, we can observe that other constants than 2 will increase the best case smoothness, too. Additionally, if we look at the longest sequences. We know from Lemma 10 that same sized ranges are adjacent. Here we can not use our general partitioning rule, so we have to establish an additional one like *in case of equality use* $1/2$ *instead of* $1/c$. Finally, only using $c = 2$ is the simplest and best we can do here.

Chapter 4

Application Area SAN, DHHT supporting RAID Architectures

Beside the domain of P2P Networks the new strategies, based upon Consistent Hashing, have entered the application area of SANs. In this chapter we present an architecture, called DHHT-RAID, usable for storage allocation in huge and dynamical Storage Area Networks. The resulting approach is based on DHHT and combines its features to overcome disadvantages of RAID caused by the needs of changing SANs.

The main advantage of this architecture is its ability to handle a dynamic set of heterogeneous devices, serving a dynamic set of data and supporting any RAID Level at the same time. In particular, the architecture uses the Linear or Logarithmic Method of DHHT and inherits all positive attributes. Furthermore, it improves its balancing quality with embedded multiple choice. In addition DHHT-RAID avoids time intensive re-stribes caused by joining or leaving devices, typically occurring in classical RAID systems.

4.1 Allocation Problems if RAID meets Random

Methods using random like hash functions and trying to support any RAID Level have to overcome some problems. The main attribute of RAID is its consecutive parallel allocation among distinct devices, which also guarantees its reliability. Related DHT based strategies commonly distribute items somehow by random and evenly among a set of nodes. This allocation strategy can break the consecutiveness of RAID allocations, but without prevention it also endangers reliability of RAID. For example, let us consider the placement of segments generated by RAID encodings independently by random among all nodes. For each block or item $d \in D$ depending on the specific RAID level, the encoding will produce a set of redundant or striping segments $D_p = \{d_1, \ldots, d_p\}$. Remember, all these segments need to be placed on distinct nodes. If the segments are distributed independently by random among V, a RAID decoding will tend to fail in some cases. This can easily be observed if devices are uniform in size. Let $f : D_p \rightarrow V$ such a random mapping function, then the probability $P[|\{f(D_p)\}| < |D_p|] > 0$ for any $0 < p << |V|$. A strategy can not ensure the reliability of RAID, if the probability of two elements in D_p sharing the same device is grater than zero. This effect is analog to the *birthday problem* for p persons, sharing the same birthday and can be declared by $P = 1 - (n!/((n-p)! \cdot n^p) > 0$. From this we can observe that the probability of such conflicts increases the less n or the greater p is. Eventually, none of such random distribution strategies can grant independent allocation patterns for each D_p. So to avoid such conflicts prevention or additional efforts during the allocation process is needed.

A possible prevention is to partition or group V into p distinct subsets. To respect RAID, each $V_i, V_j \in \{V_1, \ldots, V_p\}$ satisfies $V_i \bigcap V_j = \emptyset$, if $i \neq j$, and $\bigcup_{i=1}^{p} V_i = V$, and $c(V_i) \approx c(V_j)$. Than each V_i is utilized individually as a single virtual device using a strategy like Share, Sieve or DHHT, see [9]. Note, if the strategy is capable of handling different sized devices, then V is allowed to be heterogeneous. With these p distinct virtual devices any classical RAID level can be applied on top without conflicts. Thus it remains to keep the capacity of all virtual devices nearly equal and their address room consistent if changes occur. However, such a system can never change p without radical impacts and data allocations.

A further preventive approach to avoid conflicts is discussed later in Section 4.2.1. It describes the use of our DHHT approach in combination with classical approaches. Beyond the two staged combination, the subsequent introduced DHHT-RAID of Section 4.3.1 exemplifies the integration of RAID Levels into DHHT. These extensions resolve conflicts not by prevention, but online. It mainly integrates multiple choices defined by the model and runtime information among

a constant number of nodes. The main improvement of DHHT-RAID is its complete independency on p and the degree of capacity utilization in combination with a parallel allocation on distinct devices.

4.2 RAID Capacity Allocation Constraints

As said, the main advantage of RAID is the improved availability and failure tolerance, as well as the increased access rate. These attributes can only be guaranteed if and only if each of the p segments is placed on a distinct and independent device. If for instance RAID V is applied and the distinct placement is violated, the reconstruction is in case of failing devices is endangered. Similar effects can be observed by all other standard or nested RAID levels.

If we enforce the distinct placement of segments, we have to take a closer look on the allocable capacity of each access pattern, especially if segments are distributed randomly. Since V consists of uniform nodes and p is defined by the specific RAID level, $c(\bar{v})$ is fixed. Unless devices are not partitioned, each device is allowed to occur only once among all $\bar{v} \in \bar{V}$. This circumstance changes, if devices are heterogeneous in capacity. Than remaining capacities of $\bar{v} = (v_1, \ldots, v_p)$ is defined by $c_r(\bar{v}) := \sum_{1 \leq i \leq p} c(v_i) - c(\bar{v})$. Remember, the capacity of a pattern was p times the capacity of the smallest participating node. These capacities are combinable with remaining capacities from other patterns to increase overall utilizable storage. Naturally, only as long as at least p devices can offer none allocated space.

Further, we are aiming the utilization of the maximal capacity, at best $c(V)$ and coevally preserving the efficiency such that $|\bar{V}| = \mathcal{O}(|V|)$. Therefore, we carefully have to select patterns out of $\binom{|V|}{p}$ possible combinations. This is necessary, since storage can only be assigned once on a device. Concerning to the competitive allocations of couple of patterns, many subsets of size $\mathcal{O}(|V|)$ can not reach the maximal allocable capacity and others will. Furthermore, for some pattern sets it can be reasonable to allocate less than $c(\bar{v})$ on each pattern to allow additional combinations, which will increase the utilization.

So we need an efficient method to determine $\bar{V}$ and for each $\bar{v} \in \bar{V}$ its allocation maximum. Finally, we will see in following Section 4.2.1 that both can be done together.

4.2.1 The Sweep Line Method

The Sweep Line Method allows us to determine the maximal allocable capacity in a heterogeneous set of nodes. It determines an independent access pattern

set $\bar{V}$ in $\mathcal{O}(|V|)$. In advance, we have to define another capacity function by overloading Definition 3 and attache a further individually selectable parameter for each pattern allocating space on p devices.

Definition 27. *For $\bar{v} = (v_1, \cdots, v_p)$ with $c(\bar{v})$ restricts $c(\bar{v}, k_{\bar{v}}) := c(v) \cdot k_{\bar{v}} \mid k_{\bar{v}} \in [0,1]$ the maximal allocable capacity of $\bar{v}$ via $k_{\bar{v}} \in [0,1]$.*

This definition enables us to restrict the allocable capacity, such that we can determine a set of real numbers to maximize $\sum_{\bar{v}\in\bar{V}} c(\bar{v}, k_{\bar{v}}) \leq \sum_{v\in V} c(v)$. This restriction we use in following method and presumes that p is known and no $v \in V$ violates the constraint $max(\{c(v)\}) \leq c(V)/p$. This can only occur for at most $p-1$ devices and is easily testable. These devices must be trimmed such that the constraint holds. The constraint is essential, and if not fulfilled, we will see there exists no distribution for all $D_p \in D$ allocating $c(V)$. The following steps describe the determination of p subsets $V_1, \ldots, V_p$ of V resulting in $\bar{V}$. We start with V_j and :

1. set $j := 1$, $l := 1$, and assign to $\forall\, v \in V$ a unique index $i \in \{1, \ldots, |V|\}$

2. **if** $V \neq \emptyset$, **then** set k such that $\sum_{i=l}^{k} c(v_i) \leq \frac{c(V)}{p+1-j} < \sum_{i=l}^{k+1} c(v_i)$ **else** continue with step 6

3. **if** $\sum_{i=l}^{k} c(v_i) = \frac{c(V)}{p+1-j}$, **then** set $V_j := \{v_i, \ldots, v_k\}$, $V := V \setminus \{v_i, \ldots, v_k\}$, $j := j+1$, and $l := k+1$ continue with setp 2 **else** continue

4. partition v_{k+1} into v'_{k+1}, v''_{k+1}, such that $c(v'_{k+1}) = \frac{c(V)}{p} - \sum_{i=l}^{k} c(v_i)$ and $c(v''_{k+1}) = c(v_{k+1}) - c(v'_{k+1})$

5. set $V_j = \{v_l, \ldots, v_k, v'_{k+1}\}$, $V := v''_{k+1} \cup (V \setminus \{v_l, \ldots, v_{k+1}\})$, $j := j+1$, $l = k+1$, and continue with step 2.

6. **for each** V_j concatenate all $v_i \in V_j$ sorted over the indices to a single virtual device u_j of capacity $c(V_j) = \sum_{v_i\in V_j} c(v_i) = c(V)/p$.

7. set the address counter $a := 0$ and $a_{start} := a$
   while $(a \leq \frac{c(V)}{p} - 1, a{+}{+})$
   for each V_j determine devices defining a and thus $\bar{v} \in V_1 \times \ldots \times V_p$.
   if $\bar{v}$ will change at $a+1$ **then** set $k_{\bar{v}} := \frac{a+1-a_{start}}{c(v)}$
   if $\bar{v}$ changes at a **then** set $a_{start} := a$ and $\bar{V} := \bar{V} \cup \{\bar{v}\}$

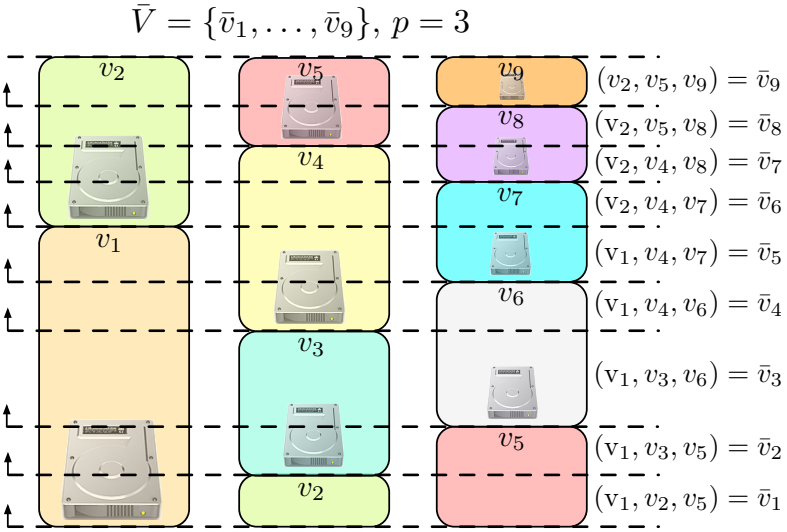


Figure 4.1: Defining $\bar{V}$ with the Sweep Line Method

This algorithm creates p subsets equal in size, defined by the concatenation of addresses gathered among the ordered indices of each $v_i \in V_j$, see step 6. To keep the order during the concatenation is essential, otherwise same addresses are potentially assigned to a partitioned device that occurs in V_j and V_{j+1}, see step 4. Lastly step 7 simulates a sweep line defining a new $\bar{v}$ each time it current address touches a new start or end address of a physical device in any V_j, compare figure 4.1. The distances of these addresses also define k_v used to restrict the capacity of $\bar{v}$. Finally, to keep the runtime of the algorithm in $\mathcal{O}(|V|)$ it will be sufficient to consider only the start and end addresses.

Corollary 15. *An environment (V, D, f) consisting of heterogeneous nodes is utilizable with a set of p wise independent patterns $\bar{V}$ with capacity $c(\bar{V}) = c(V)$ only if*

$$\max(\{c(v)\}, v \in V) \leq \frac{\sum_{v \in V} c(v)}{p}$$

So the Sweep Line Method always determines $\bar{V}$ reaching the maximal p wise independent allocable capacity. However, if it reaches $c(\bar{V}) = c(V)$ only depends on the necessity of trimming devices. If this is left out but necessary, the method will define an access pattern $\bar{a}$ which is not independent.

4.3 DHHT Supporting RAID Level

In this section we describe how RAID goes along with DHHT. Foremost we will consider a further opportunity, considerable as a two stage or level approach. It combines the Sweep Line Method of Section 4.2.1 with any of the DHHT methods of Section 2.2. The second and main architecture, we denote as DHHT-RAID, is an enhancement of the Linear or Logarithmic DHHT method. It supports any kind of RAID Level simultaneously. Further, it manages the additional incidental meta data in its own data structure. Finally, we will show the performance of DHHT-RAID with some simulation results.

4.3.1 DHHT Featuring RAID

As mentioned in Section 4.1 one solution is to group V into p distinct sets and run on top any specific RAID level. However, with the previous results, the DHHT and the Sweep Line Method, we can describe an alternative combined solution. The basic steps are the following

1. check whether any $v \in V$ violates $c(v) \leq \frac{\sum_{v \in V} c(v)}{p}$. If yes, mask out the overcapacity of according devices such that the constraint holds.
2. apply the Sweep Line Method according to the RAID level with the adapted V and determine $\bar{V}$.
3. apply any DHHT Method and use $\bar{V}$ as input. Use each $\bar{v} \in \bar{V}$ as node and according to the chosen DHHT method choose $c(\bar{v}, k_{\bar{v}})$ as weight within the distance function.
4. distribute all $D_p \in D$ among M as items. Store the encoded segments of D_p on the distinct devices behind the chosen $\bar{v}$ minimizing the hight.

This architecture simply uses DHHT and its ability to utilize any set of dynamic heterogeneous node set. Thus all attributes form DDHT can be observed here to. Clearly, DHHT can be substituted with any other method capable of handling such a heterogeneous set environment like Share or Sieve. The guarantee of finding distinct places for segments is given by the Sweep Line Method. Thus it only remains to resolve the heterogeneity of access pattern capacities, which all these methods perfectly can do. However, dynamics in this architecture compensated by DHHT only consider join and leave operations of any $\bar{v} \in \bar{V}$ and not of any $v \in V$.

So let us first consider the join operation in V. First, if we insert a single node and then redetermine $\bar{V}$, each $\bar{v}$ will at least have an adapted $k_{\bar{v}}$ and the costly

re-stribes are as intensive as in the classical static approach. The advantage of this architecture can be seen if we insert a further set of nodes V'. Additionally, nodes with remaining trimmed capacities from previously applied Sweep Line Methods can be joined with these capacities in V'. With V' we do the same as we have done before with V. The important part is that the followed insertion of all $\bar{v} \in \bar{V}'$ into DHHT will rebalance all $\bar{v} \in \bar{V} \cup \bar{V}'$ with the costs of DHHT. Note, if other methods than DHHT are used, according to their costs. Eventually, we can merge any independently generated $\bar{V}$ together, if they were built for the same RAID level. The disadvantage of this method is, we can only extend the capacity if V' consists of at least p devices and for no $u \in V'$ the constraint $u \leq c(V')/p$ is violated.

A leaving operation of a node $v \in V$ followed by a join of u replacing v, where $c(u) = c(v)$ has no impact. If $c(u) > c(v)$ then $c(u) - c(v)$ remains unused, unless it can be integrated again with remaining capacities. Basically, a leaving node has an impact on all $\bar{v} \in \bar{V}$ it is participating at. If we only want to lose the capacity of $c(v)$, the favorable case is, if no node participates at involved patterns with a capacity larger then $(p-1) \cdot c(v)/p$. Then we can determine with these nodes a further pattern set V' without remaining unused capacities. Otherwise we also lose the capacities remaining after the accomplishment of the Sweep Line Method. Furthermore, if only $p-1$ nodes are involved and left we will lose the capacity of $p \cdot c(v) \leq c(V)$. Additionally, if a leaving node is of size $c(V)/p$, $\bar{V}$ must be redetermined completely and all items must be distributed again, because no valid $\bar{v}$ left. If we consider the item movements, we first have to move at most $p \cdot c(v)$ space by the removal of involved patterns. Second if the remaining capacities can be used completely, its integration will cost at most $(p-1) \cdot c(v)$ induced by the rebalancing of DHHT. Whereas an optimal strategy will only have to move the items stored on v.

Corollary 16. *With a given p, extending $\bar{V}$ by joining $V \cup V'$, results in best case to a capacity extension of $c(V') + c(V) = c(V \cup V')$ with costs of DHHT or a substitute. In worst case the extension is impossible. In best case reducing $\bar{V}$ by setting $V := V \setminus \{v\}$, we lose $c(v)$. In worst case we lose $p \cdot c(v) \leq c(V)$. In case of losing $c(v)$ we have to move at most $(2p-1)c(v)$ space, so we are $2p-1$ competitive.*

So far we can state out that the flexibility of this or other multi stage solutions is not satisfactory. In both cases of dynamics we have a strong dependancy on p inducing additional costs or unsolvable constraints induced by the capacity distribution. The solution introduced in the following tries to overcome this dependancy by combining runtime information with DHHT structures. It combines multiple choices with the determination of allocation patterns, by the costs of additional meta data for each item, to fight against these intrinsic deviations.

4.3.2 DHHT with Embedded RAID, DHHT-RAID

In this section we introduce an architecture capable of resolving heterogeneity of dynamical node sets and coevally supporting any RAID level. In contrast to the previous section it overcomes costs induced by the dependancy on p by using additional structural information from the Linear or Logarithmic Method of the DHHT model.

As we have seen in Section 2.2 the coverage of the hash range M is always guaranteed. If a node is removed, its successor is already defined by the fact that it defines the 2nd minimum at the same position. This allows us to resolve one major constraint of RAID, the need of distinct places for each item segment. So if we use this attribute straight forward, we can describe the following. If p is defined by the RAID level we have to select p distinct nodes. So we choose a random position r_{D_d} in M and the node v_1 defining the minimal hight at r_{D_d} receives the first segment of $D_p = \{d_1, \ldots, d_p\}$. By the requirements of RAID the receipt of d_1 disables v_1 to participate in further distributions of D_p. Thus, we would have to remove the node to exclude the chance of further $p-1$ assignments. Instead of choosing $p-1$ additional positions for the remaining segments we simply reuse r_{D_p}, but without v_1. All in all we do this p times, where consecutively nodes which have already received a segment are kept disabled. Essentially, this is the same as if we take at r_{D_p} the sorted order of heights among V and select the first p with the the smallest hight. On these nodes $\{v_1, \ldots, v_p\}$ we store the segments of D_p. A simple example of this distinct node selection defining $\bar{V}$ is pictured in Figure 4.2 by the Linear Method of DHHT. Note, similar to the Sweep Line Method, each $\bar{v}$ is associated with a range $I_{\bar{v}} \in M$ and $c(\bar{v})$ with its length $|I_{\bar{v}}|$. In contrast,

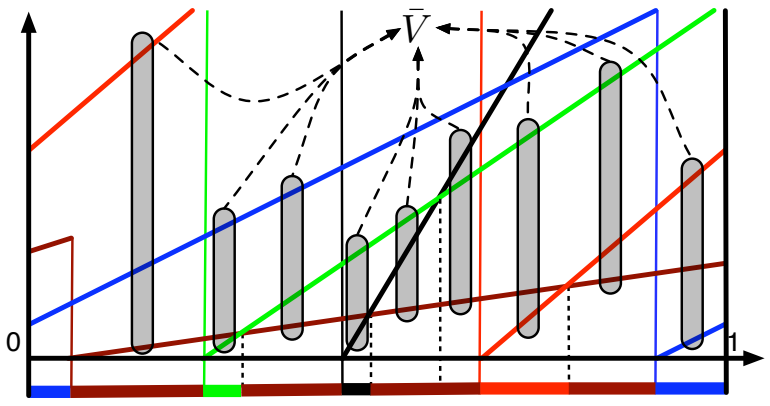


Figure 4.2: $\bar{V}$ induced by Linear DHHT minimum with $p = 3$

because of DHHT and its fragmentation structure, $\bar{v}$ can occur more than once in M. Thus the expected capacity of $\bar{v}$ is reflected by all ranges associated with $\bar{v}$.

Let us now consider dynamical changes in V. Similar to Section 2.2.2, where only I_{min} was modified, we require an assignment verification. The difference is that we now have to observe the first p functions at each position in M. If the mapping order of items among the p nodes matters, we have to respect the order changes caused by node fading, see Section 2.3.2, too. In the following we consider the insertion and removal of a single node. Additionally we assume that the mentioned order does not matter. We say v dominates u at a position r_d or within $I_{\bar{v}}$ if the hight, according to the DHHT method, of u is less than the hight of v. First, the insertion of v. We have to determine the ranges with their associated patterns, where v would dominate i nodes, where $0 \leq i \leq p-1$. For every involved pattern $\bar{v}$ there exists a node v' previously dominating $p-1$, and p nodes after the insertion of v. For each of these $\bar{v}$ and D_p, where $r_{D_p} \in I_{\bar{v}}$ we have to move the segments stored on v' to v. Second, if v leaves the environment, we have a similar situation. The node v has to transfer after its removal all segments $f^{-1}(v)$ to the nodes now dominating exactly $p-1$ others. The mapping is analog to the insertion. Important is that independent of p and thus the according RAID level, the insertion, removal or change of a single node implies the movement of at most one $d_i \in D_p$ for all $D_p \in D$. Which leads to the following corollary.

Corollary 17. *For a given p, a join operation is monotone and 1-competitiv if v receives $\frac{c(v)}{c(V \cup \{v\})} \cdot |D|$ items. Assuming v stores $\frac{c(v)}{c(V \cup \{v\})} \cdot |D|$ items, then a leave operation is monotone and 1-competitiv.*

This corollary is not very strong, because we can not state out the balancing attribute for a single node. Remember, the analyses we have made for DHHT are considering the assignment of one item at a time. By reusing a random position and changing the environment we can not stat out any theoretical result about balancing qualities. By changing the model we have obtained strong dependancies.

Our aim is to maximize the allocation of $c(\bar{V})$, compare Figure 4.2, but during the distribution some patterns or nodes will have no capacity left before others. Independent on this, they are still offering themselves within M. Thus, a further segment assignment on these nodes is not feasible and requires additional effort to allow further item assignments. Our idea is inspired by multiple choices and especially by the power of two choices [30]. So we include a further balancing based on runtime information to keep the deviation from the desired share of nodes low. Conventional multiple choice processes, for instance in DHTs, define a certain number of k randomly chosen positions in M and take the best one as final position. So each $v \in V$ is involved and potentially chosen at most k times.

However, for each assignment of $d \in D$ we have to store the information which of the choices we have made. Otherwise every time we want to access d we have to look at k nodes. Typically these k positions are generated by using k hash functions with corresponding attributes. So if we want to ask only one node, we have to store for each item the information which function we have chosen. Our idea goes a slightly different way.

In contrast to the conventual multiple choice we determine the choices with a single random position. Therefor, we adapt the previous scheme. Instead of choosing the first p nodes at r_{D_p} we take the first $p+k$ nodes at r_{D_p}. This leads to the architecture, exemplified in Figure 4.3, which needs the following steps to distribute D among V supporting any RAID level.

1. setup the environment according to the chosen Linear or Logarithmic Method. Determine k and p according to the RAID level.
2. for each $D_p \in D$ determine a random position $g(D_p) \in M$.
3. choose at r_{D_p} the first $p+k$ nodes without their copies, denoted by $barv_{p+k}$.
4. select those p nodes from the $p+k$ in $\bar{v}_{p+k}$ which minimize $c(f^{-1}(v))/c(v)$, in case of equality choose the node minimizing the hight according to the chosen method. These nodes are defining $\bar{v}$.

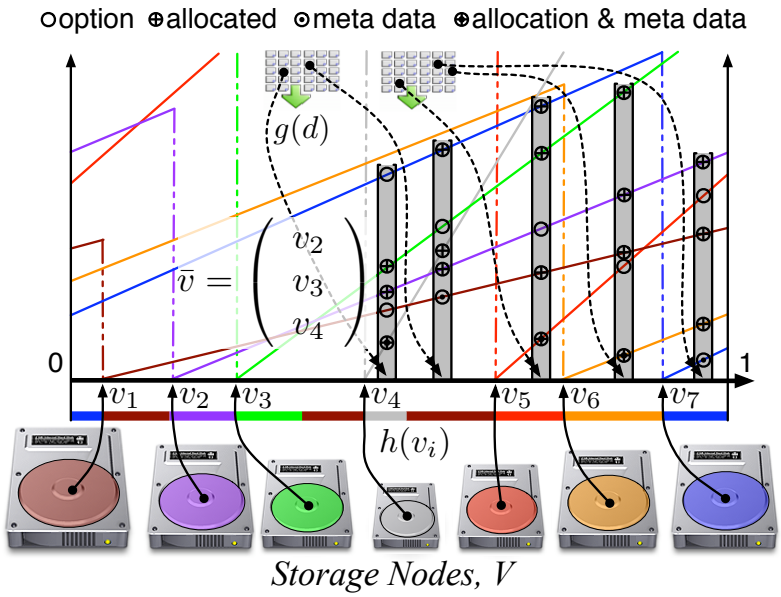


Figure 4.3: DHHT with embedded RAID and Multiple Choice

5. distribute all $d_i \in D_p$ among the nodes identified by $\bar{v} = \{v_1, \ldots, v_p\}$.
6. define a meta ticket T_{D_p} at least storing $\bar{v}$ and the physical address of each d_i on v_i.
7. store T_{D_p} on v if $r_{D_p} \in I_v$ and $I_v \in I_{min}$ according to the chosen method.

Remember, both DHHT variants need a sufficient number of copies to achieve their balancing quality. So step 3 is important to avoid multiple choices of same nodes, otherwise it would endanger the independency of $\bar{v}$. A further note to the balancing. If the value of $\max(\{c(d)/c(v)\})/\min(\{c(d)/c(v)\}), d \in D_p$ is huge, the single allocation on small nodes will have an huge impact on the percentage load. Thus, it would be better to use $\frac{c(f^{-1}(v))+c(d)}{c(v)}$. This will disburden smaller nodes and prefer bigger ones, because it respects the node level after a potential allocation.

Since the original DHHT only identifies the nodes, we now can use the meta ticket T_{D_p} for each $D_p \in D$ and store additional information within the environment. For example this might be the encoding of D_p, even p can be stored and thus possibly changed for each D_p individually. Furthermore, because T_{D_p} is placed as usual in DHHT, the meta information will be distributed proportional among the node capacities. Beside this we can also observe that meta information and data is possibly stored on distinct nodes. So we have merged the data and the meta information systems into a single model to ease requirements of central structures.

Finally, in case of dynamical changes we can reuse the mapping idea we have described without choices. If a node u joins we have to take all $\bar{v}_{p+k}$, u now participates at. For each range $I_{\bar{v}_{p+k}}$, u takes segments from the evicted nodes. According to the heterogeneity of nodes the workload of u possibly exceeds the workload of any node in appropriate $\bar{v}_{p+k}$ not storing segments of D_p. So u takes items until this occurs and remaining segments of each D_p are distributed balanced among the other nodes not storing any segments of D_p. Analog if a node leaves, then we distribute the segments balanced among the $k+1$ nodes not storing any segments of D_p. All changes and the necessary information can be deposited in T_{D_p} also. The meta tickets migrate as defined by I_{min} determined by the chosen DHHT Method.

4.3.3 Simulation Results

In this section we present some simulations picturing the performance of DHHT-RAID. We focus on the weakest strategy, the Linear Method used with a different number of virtual nodes, multiple choices, and parallel allocations, as proposed

in Section 4.3.2. Further, we will show some results, the random node position sampling is substituted by our deterministic decomposition Φ-Div. ***In contrast to the our previous results we do not use partitions as recommended.*** The simulation results will show that the embedded balancing is an appropriate substitute. The simulation was implemented in java. Further, the random positions have been determined by the Mersennen Twister included in the RngPack-1.1 for java. The seed value for all experiments was equal.

The Experiment and Testing Environment

As we have said our main application area is some kind of Storage Area Network, where participating nodes are equipped with a fixed capacity. So it is of tremendous importance that a node does not receive more allocation requests than it is capable of storing. If a strategy can not avoid this, an additional managing overhead is needed to continue allocations. Each node dropping out this way needs a special unwanted treatment concerning write and lookup operations for items. Because of this, our experiment procedure is the following. We first setup the specific environment according to the method and the given node weight distribution. Then ***we throw one item after another until the first node drops out***, because of over allocation. In case of parallel allocation the experiment ends if one of the segments in D_p is unplaceable. So, in best case the experiment ends because all nodes have reached their limits. However, in worst case the imbalance of a single node leads to a very early experiment abort.

We have used the following heterogeneous environment to run our experiments.

- V consists of $16.389 = 2^{14} + 5$ nodes.
- The capacities are distributed evenly among V out of $\{10, 40, 80, 100, 160, 250, 500, 750, 1000\}$.
- The weight unit of nodes are GigaBytes and we distribute items of size $c(d) = 100$ MegaByte. Thus the overall capacity of the enviroment $c(V) =$ 5.262.690 GigaByte.
- In case of parallel allocations, to simulate RAID, the item capacity is $c(D_p) = p \cot c(d)$.

DHHT-RAID performace

In the following you find some table figures picturing the utilization performance of DHHT-RAID. We first show the basic performance with balancing but without parallel item allocation and afterwards in combination with parallel allocation.

The table figures and their embedded images showing the experiment results are explained as follows. Each image rectangle describes a single experiment with specific parameters. The coordinate within the table identifies the used parameter range given in the caption. In a single experiment image the y-axis denotes the single node level in percent and the x-axis the over all allocated system capacity. The color of each rectangle point describes how many nodes of $|V|$ are having the same load, in percent. Here gray means no node has this load, blue means at least one node or a few are having the same load. The color transition from blue to red to yellow shows the increasing node density where nodes are having the same load. The color transition coding is also given in Figure 4.4. Finally, the white rectangle area shows the remaining not allocated capacity of $c(V)$ because of an early experiment abort.

The diagonal of each rectangle describes the desired share of a node during any experiment. So nodes will paint traces like a comet trail exemplifying the deviation concentration around the diagonal. If the experiment aborts, because any node reaches more than 100% the painting stops and the remaining area is left white. However, the closer these traces are to the diagonal, the better the experiment is and the more capacity $c(V)$ we can allocate. If this is the case the size of the white surface will decreases. Thus the, aim is to catch those nodes painting trails with the highest or lowest trace gradient and concentrate them over the diagonal. So the best we can expect is a completely flooded gray rectangle with a yellow diagonal from the left bottom to the upper right top and no white surface.

Figure 4.5 shows the allocation behavior of experiment runs with random node positions, node copies between $0 \leq C \leq 2$, a balancing between $0 \leq B \leq 5$ position for a single item, thus no parallel placements $P = 0$. Figure 4.6 shows experiment runs similar to Figure 4.5, but copies between $3 \leq C \leq 5$. Experiments considering further values for copies can be found in Figure 4.7 and Figure 4.8.

Figure 4.9 shows the allocation behavior if random node position sampling is substituted by a deterministic Φ-*Div* decomposition. We have left out some

Figure 4.4: Linear Scaled Color Legend of Node Density;left $> 0\%$ right $\leq 100\%$

values for copies, because the tendency is observable without them. These simulations are also running without parallel allocation and a balancing between $0 \leq B \leq 5$.

The Figure 4.10 gives the overall capacity which we were able to allocate before the first node aborts in percent. The value corresponds to the fraction of the gray area compared to the whole rectangle size of the images in the previous Figures. Figure 4.11 shows this too, but with an increasing number of items placed in parallel on a single position r_{D_p}. We do not list the images showing the allocation behavior of these experiments, because they look the same and because we are only interested in the impact on $c(\bigcup_{v \in V} f^{-1}(v))$.

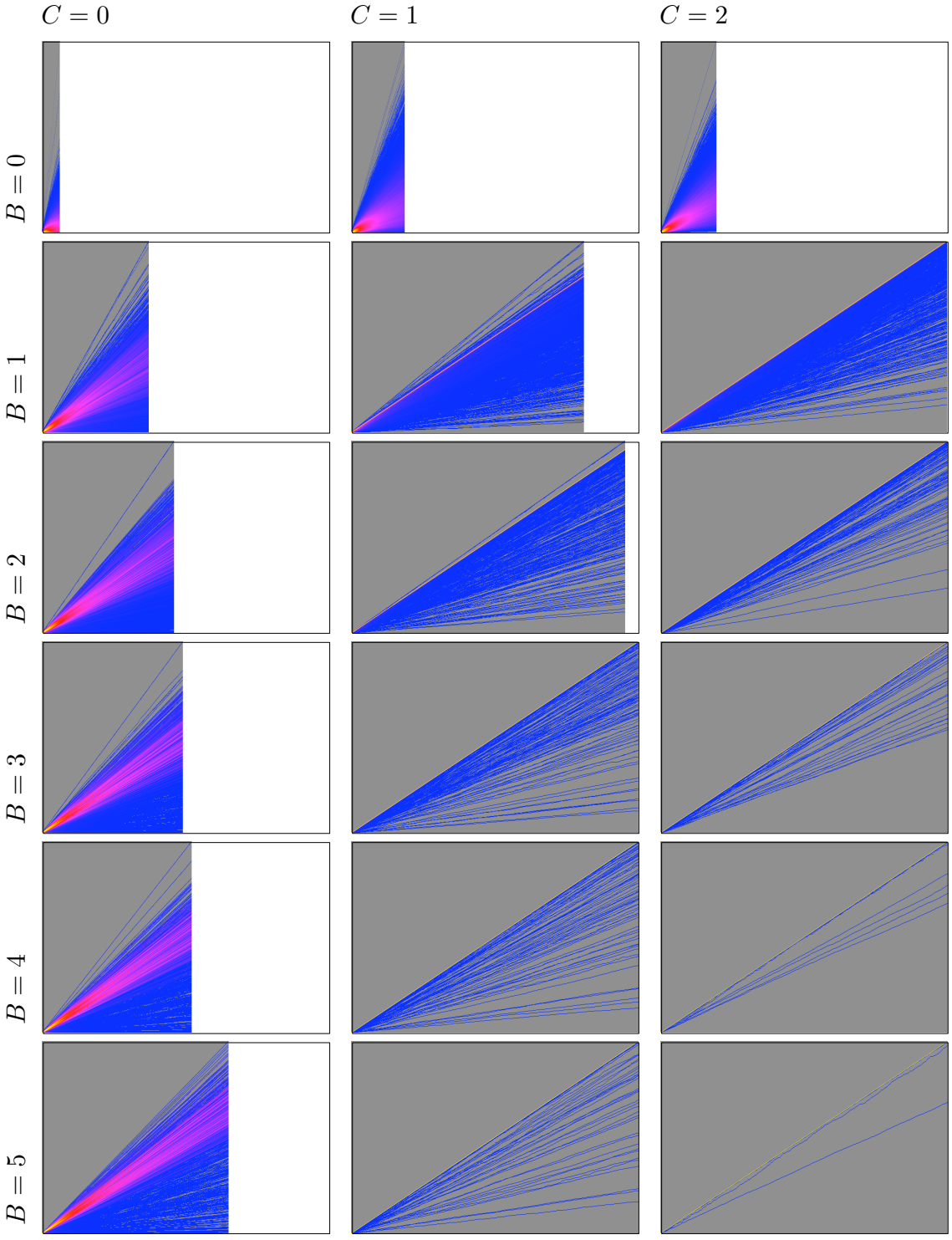


Figure 4.5: Random Positions, $P = 0, 0 \leq C \leq 2, 0 \leq B \leq 5$

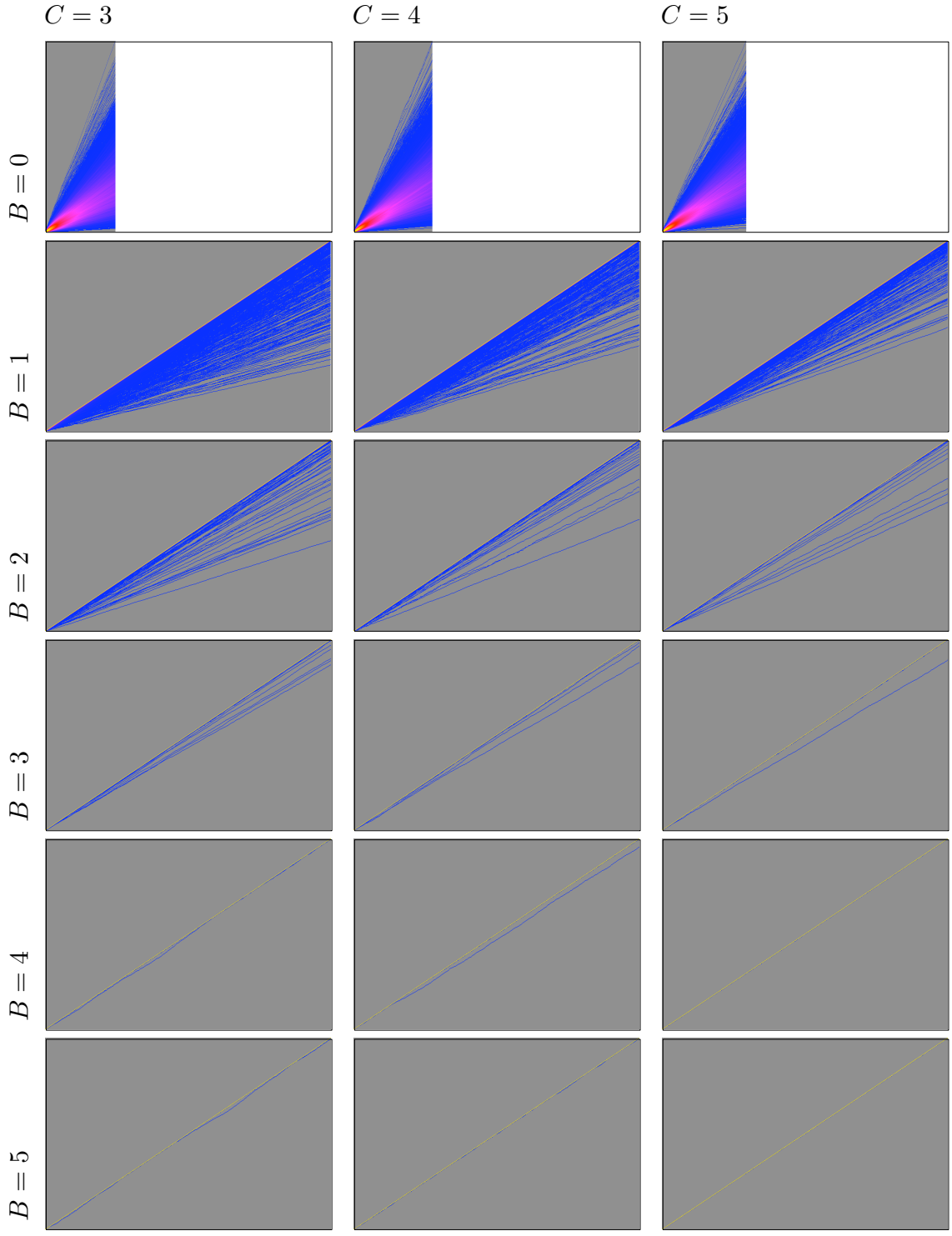


Figure 4.6: Random Positions, $P = 0, 3 \leq C \leq 5, 0 \leq B \leq 5$

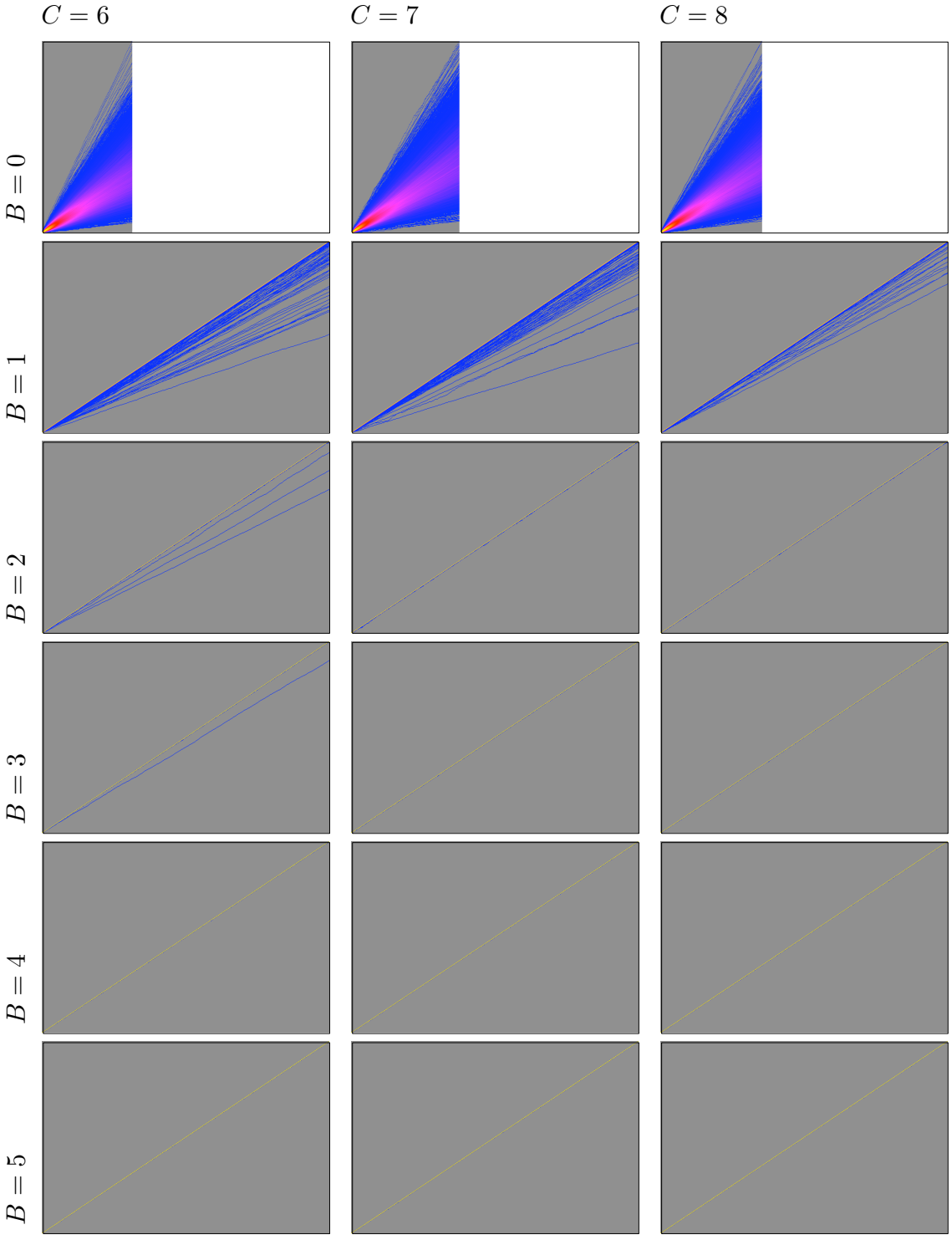


Figure 4.7: Random Positions, $P = 0, 6 \leq C \leq 8, 0 \leq B \leq 5$

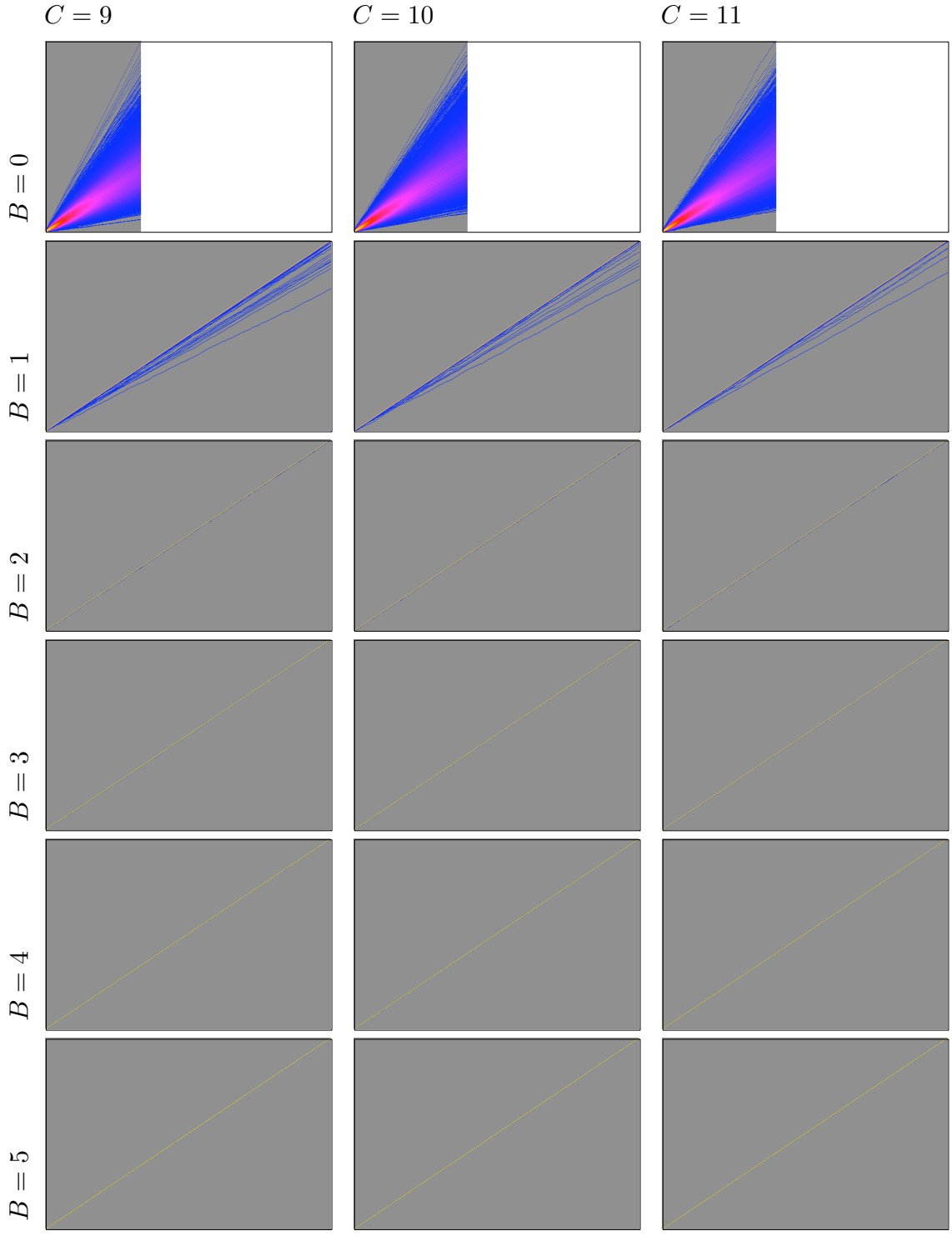


Figure 4.8: Random Positions, $P = 0, 9 \leq C \leq 11, 0 \leq B \leq 5$

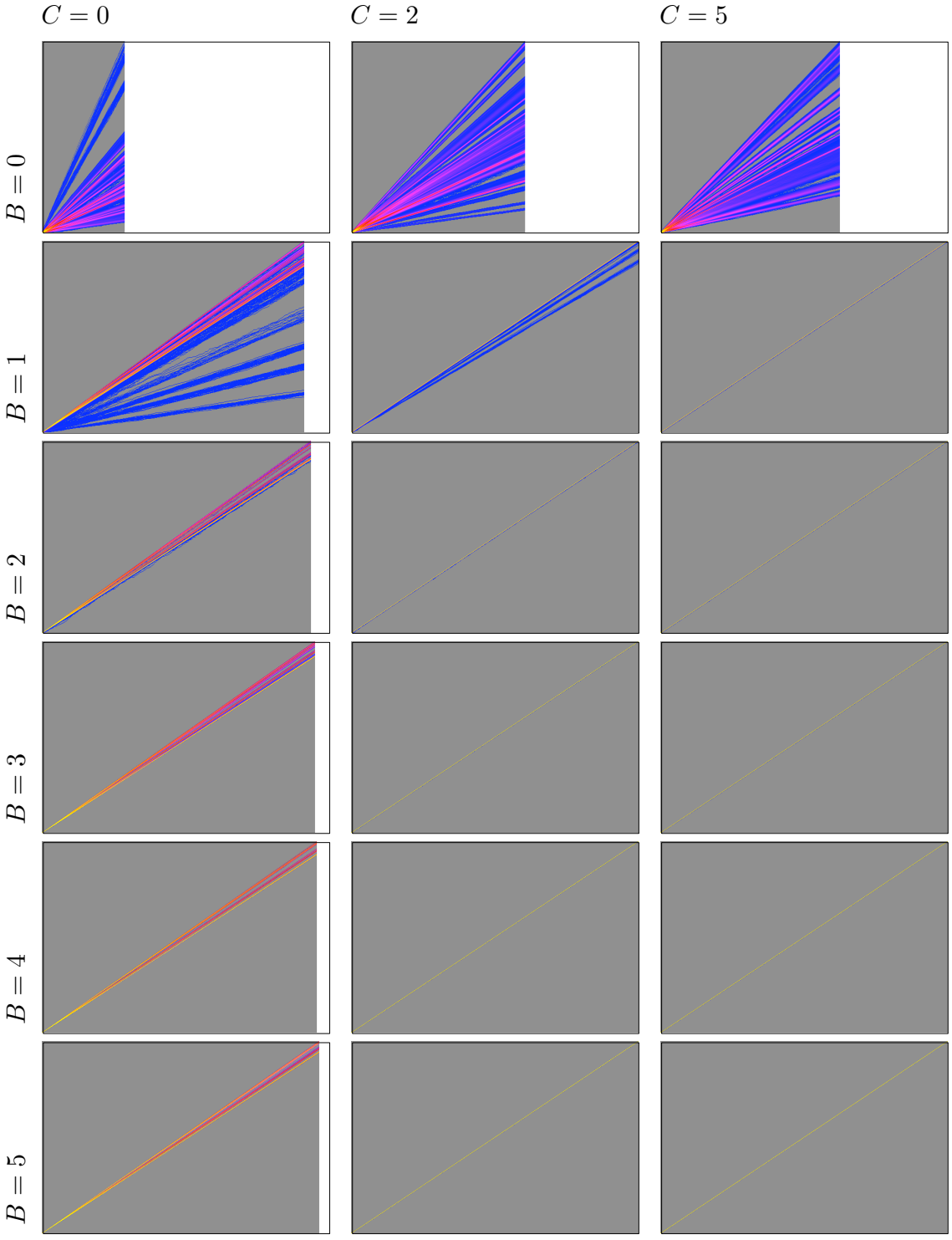


Figure 4.9: Φ-*Div* Positions, $P = 0, C \in \{0, 2, 5\}, 0 \leq B \leq 5$

allocation experiments, where r_v is chosen by random						
% of $c(V)$	$B = 0$	$B = 1$	$B = 2$	$B = 3$	$B = 4$	$B = 5$
$C = 0$	6.2333	37.1917	45.9875	49.0520	52.1087	64.9623
$C = 1$	18.6507	80.9921	95.3292	99.9315	99.9684	99.9848
$C = 2$	24.6183	99.7195	99.9617	99.9883	99.9972	99.9982
$C = 3$	24.6081	99.8681	99.9821	99.9972	99.9985	99.9988
$C = 4$	27.5971	99.9401	99.9946	99.9980	99.9990	99.9991
$C = 5$	29.4708	99.9656	99.9940	99.9982	99.9982	99.9991
$C = 6$	31.4426	99.9822	99.9966	99.9975	99.9990	99.9991
$C = 7$	37.6550	99.9869	99.9968	99.9981	99.9983	99.9990
$C = 8$	34.7315	99.9923	99.9949	99.9984	99.9987	99.9991
$C = 9$	33.5573	99.9926	99.9949	99.9984	99.9987	99.9990
$C = 10$	39.8832	99.9921	99.9969	99.9987	99.9959	99.9988
$C = 11$	39.9737	99.9942	99.9970	99.9988	99.9991	99.9992
allocation experiments, where r_v is chosen by Φ-*Div*						
% of $c(V)$	$B = 0$	$B = 1$	$B = 2$	$B = 3$	$B = 4$	$B = 5$
$C = 0$	28.8147	91.2449	93.6124	95.0020	95.6645	96.4933
$C = 2$	60.5131	99.9805	99.9966	99.9974	99.9985	99.9988
$C = 5$	62.4495	99.9941	99.995	99.9969	99.9982	99.9985

Figure 4.10: $c(\bigcup_{v \in V} f^{-1}(v))$ from Fig. 4.5, Fig. 4.6, Fig. 4.7, Fig. 4.8, Fig. 4.9

Since most embedded table images can only give an impression on distribution behavior, additional magnified illustrations can be found in the Attachment in Chapter A, for each experiment, where a node deviates significantly from its desired share. Additionally, we show for each listed experiment the final node level distribution among the different node weights after the experiment abort.

The major conclusion we can draw from these readings is, that the node deviation seems to spread similar over all weights and nodes deviate only by a constant factor. This can be argued by the linear traces nodes are painting during the experiment. Further, we can see that increasing balancing catches first nodes decreasingly sorted among their weights. This can be explained by the representation of nodes and their distance to an appropriate position r_{D_p}. However, by choosing an sufficient number of copies we can shorten the distance of nodes to each other. Especially, the substitution of random node positions by Φ-*Div* keeps the domination of small nodes by larger left neighbored nodes low. The deterministic choice avoids too short distances, because of the fixed segmentation ratio. Further, small nodes can not be dominated too early or become too huge, which also explains the better results if balancing is not used.

allocation experiments, where r_v is chosen by random				
% of $c(V)$	$P = 1$	$P = 3$	$P = 5$	$P = 7$
$C = 2, B = 5$	99.9983	99.9846	99.95622	99.9223
$C = 3, B = 4$	99.9986	99.9914	99.9655	99.9311
$C = 4, B = 3$	99.9984	99.9895	99.9528	99.9005
$C = 5, B = 2$	99.9988	99.9781	99.9067	99.8232
$C = 8, B = 1$	99.9968	99.9748	99.8662	99.7457
allocation experiments, where r_v is chosen by Φ-Div				
% of $c(V)$	$P = 1$	$P = 3$	$P = 5$	$P = 7$
$C = 2, B = 5$	99.9988	99.9974	99.9963	99.9947
$C = 3, B = 4$	99.9957	99.9949	99.9867	99.9877
$C = 4, B = 3$	99.9984	99.9945	99.9915	99.7823
$C = 5, B = 2$	99.9978	99.9950	99.9866	99.9786
$C = 8, B = 1$	99.9973	99.9938	99.9895	99.9724

Figure 4.11: Parallel Allocations, Capable to Serve RAID

The most important observation we can make for both variants is the following. By combining copies and balancing among distinct choices we can decrease the node deviation enormously. Further, we can find parameters below $log|V|$ for both to catch all dropping nodes. This behavior is still present if we allocate items in parallel as required by RAID and proposed by DHHT-RAID. This keeps DHHT-RAID efficiently in allocation runtime and space complexity. Especially the need of partitions as proposed in the Linear Method of DHHT can be compensated by multiple choices. Hence the Logarithmic Method is capable to run without out partitions, we can expect a similar behavior and possibly better results with the same parameters. Finally, the results imply that we need less copies if we embed multiple choice. Further, we need less copies and less balancing if we choose node positions deterministically, which avoids early or late dominations of nodes in DHHT.

Chapter 5

Conclusion and Outlook

In this chapter we summarize our achievements discussed in the previous chapters and consider unsolved problems of our main application area SAN.

In this work we have considered the allocation of resources among a set of nodes V by a set of items D. The focus of our strategies was the balanced allocation of limited node resources, like capacity offered by heterogeneous nodes connected via unspecified network. The challenge of these distribution techniques is to cope dynamical changes in V and D, especially if V consists of heterogeneous nodes and to preserve the desired distribution properties coevally.

5.1 What have we done

In Chapter 2 we have seen that the main afford in distributed resource allocation was made to resole heterogeneity of a single node attribute. Under the assumption that distributed units in D are uniform we believe that a fair relative allocation is reasonable and leads to good results. We have considered resolving heterogeneity by uniform solutions and strategies like Share or Sieve. Beside the mentioned solutions our introduced method has additional features. First, the coverage of the hash range is always guaranteed, independent on any kind of changes in the node set V. Furthermore, the representation of nodes within DHHT is completely independent. This unique improvement is crucial to avoid model adaption if many environment changes are occurring.

In Chapter 3 we have introduced a deterministic decomposition of a unit range. The idea of this approach is to work as substitute in classical DHT approaches. The benefit of the Φ-Div^+ algorithm is its low worst case smoothness and the low constant number of different sized intervals. This low fragmentation during dynamical changes is obtained by the consistent use of the golden ratio and some additional tricks. However, this is not the best one can reach, but the constant number of different sized intervals keeps the organization overhead small. In case of node insertion or removal we only have to search for one interval out of a group of intervals with a certain size.

In Chapter 4 we have combined all features of the Linear and Logarithmic DHHT method, to support widely used RAID Levels. Additionally, inspired by multiple choices, we have described how to apply within DHHT a balancing method among nodes. We also have described the distribution of additional meta data storing information about the parallelism. Further, we have seen in Section 4.3.3 simulation results of a simplified DHHT method in combination with parallelism and load balancing by involving a constant number of nodes at a single position. Furthermore, we have seen that a substitution of the random node sampling with a deterministic approach Φ-Div leeds earlier to less deviation among the nodes, considering the number of additional copies and balancing positions.

5.2 What is left to do?

All seen strategies have in common that they can resolve the heterogeneity of a single node attribute only. Beside this, some methods of the DHHT model are capable to support RAID and guarantee its requirements, but other environment characteristics are still left unconsidered. As we have motivated, items or tasks in D are typically not uniform in size. Additionally, because of popularity, items are typically different in access frequency. So we have to improve the understanding of heterogeneity of items. Furthermore, nodes offering storage capacity are different in bandwidth or other cumulative latencies. So one can imagine that fairness according to capacity allocation and distribution can lead to a swamped and congested environment, too. Therefore, we simply have to increase the popularity of items stores on huge but slow nodes steadily. So the future challenge might be a more complex combination of item and node heterogeneity by finding autonomous adapting strategies, combining multiple attributes and different types of dynamics, but coevally minimizing the delivery time of items or the processing time of tasks with respect to their importance.

Chapter 6

Acknowledgments

This is by far the shortest and easiest chapter, but it was fundamental to start and finish this work.

I would gratefully thank Prof. Dr. Friedhelm Meyer auf der Heide for his guidance, and patience, and most notably for giving me the opportunity to make my steps in the researchers world. Furthermore, I would like to thank Prof. Dr. Christian Schindelhauer for his joint researching work, keeping my direction during this learning process.

Finally, I say 'Thank You' to everyone who has supported me during this period and who has to stand my flaws ;-)

Appendix A

Magnified Illustrations and Additional Result Figures

In this Chapter we attach experiment illustrations previously used in compressed table views of Section 4.3.3. We show only those runs, where at least a single node deviates significantly, visible as blue trace over gray. Further, for each listed corresponding experiment we show the final node level distribution. The notation is the same as before in Section 4.3.3, C denotes the number of additional virtual nodes aka. copies, B denotes the number of additional choices for places, and denotes P the additional number of items allocated in parallel on distinct nodes at one item hash value r_{D_p}. As a reminder, the color coding of each rectangle point describes nodes with same deviation, in percent over all nodes, given in Figure A.1.

Figure A.1: Linear Scaled Color Legend of Node Density; gray$=0\%$, left $> 0\%$ right $\leq 100\%$, white=no value

The diagonal also describes the desired share of each node during the experiment.

Figure A.2: From Fig. 4.5 with $P = 0$, $C = 0$, $B = 0$

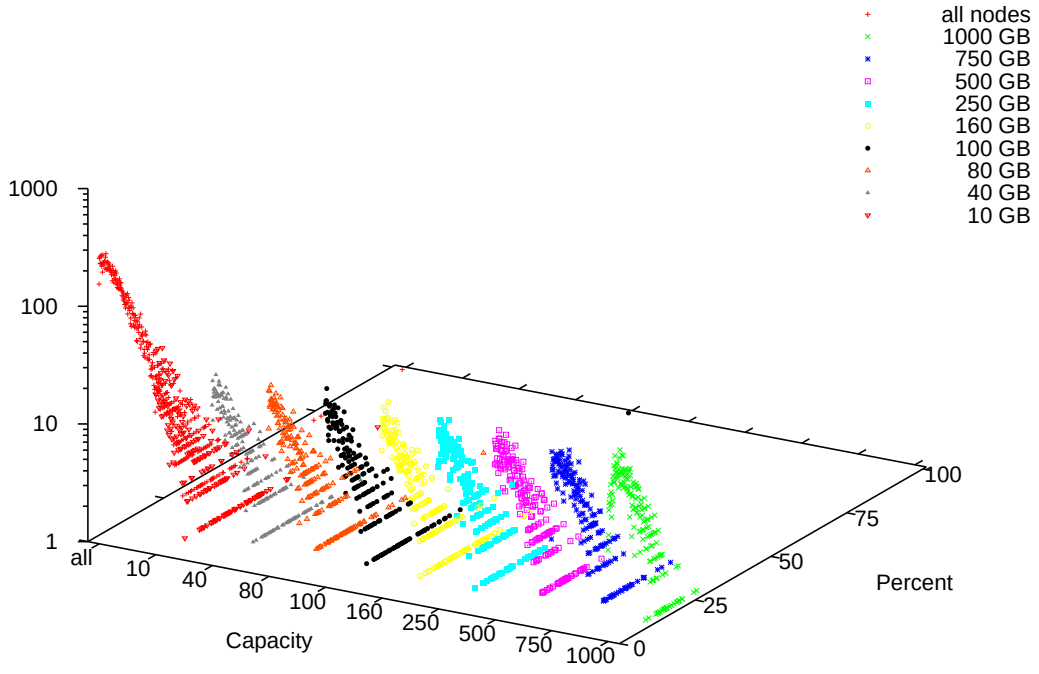


Figure A.3: Node level distribution of Fig. A.2

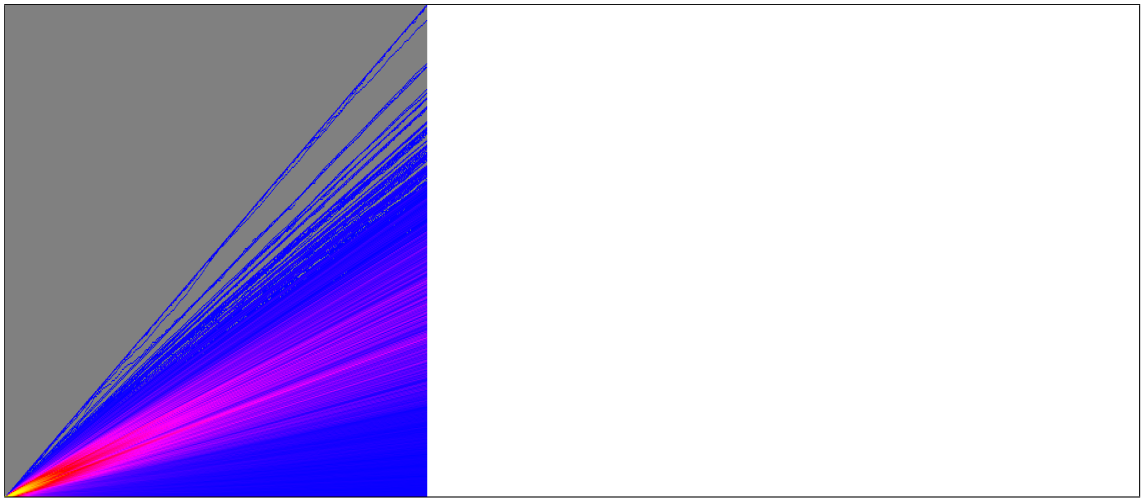


Figure A.4: From Fig. 4.5 with $P = 0$, $C = 0$, $B = 1$

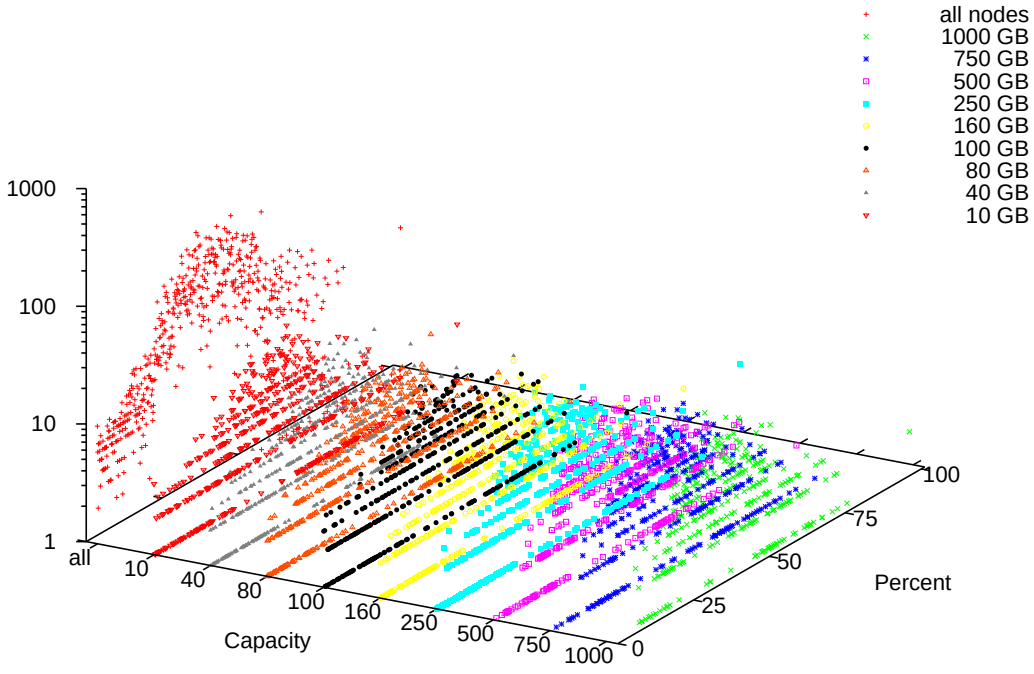


Figure A.5: Node level distribution of Fig. A.4

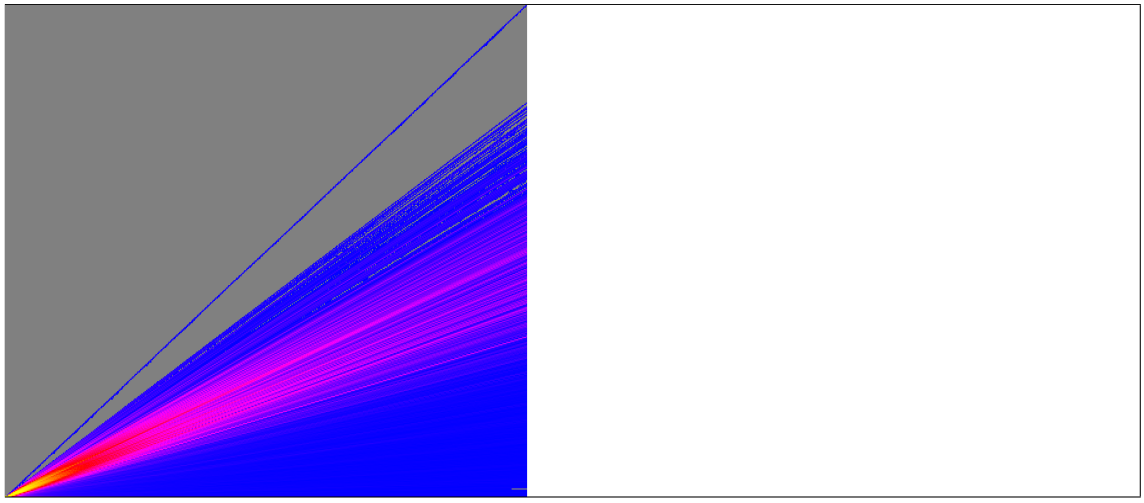


Figure A.6: From Fig. 4.5 with $P = 0$, $C = 0$, $B = 2$

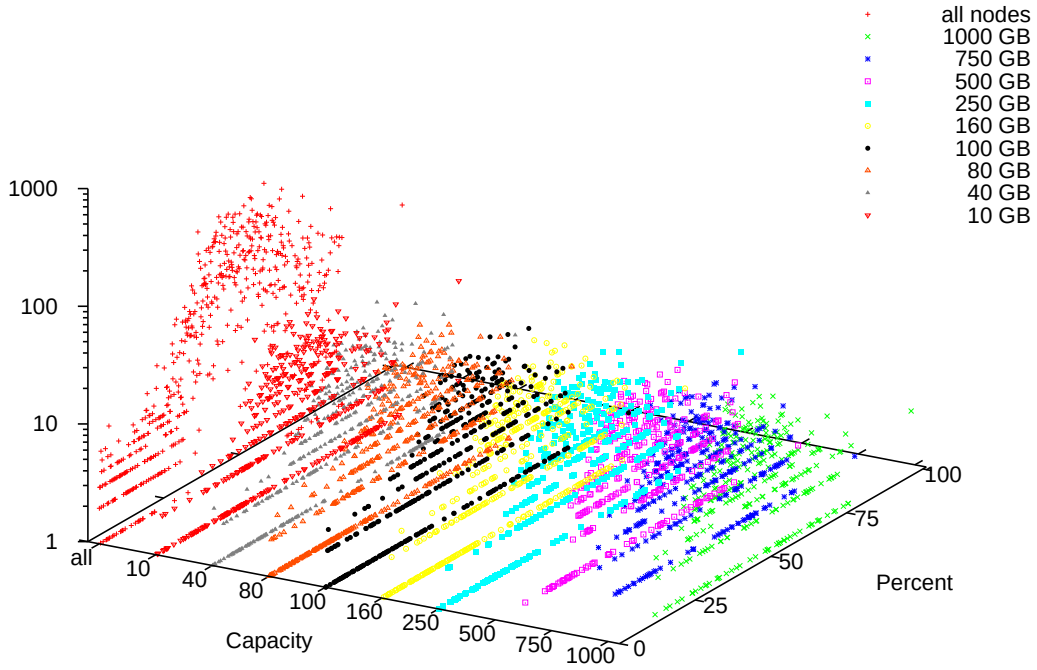


Figure A.7: Node level distribution of Fig. A.6

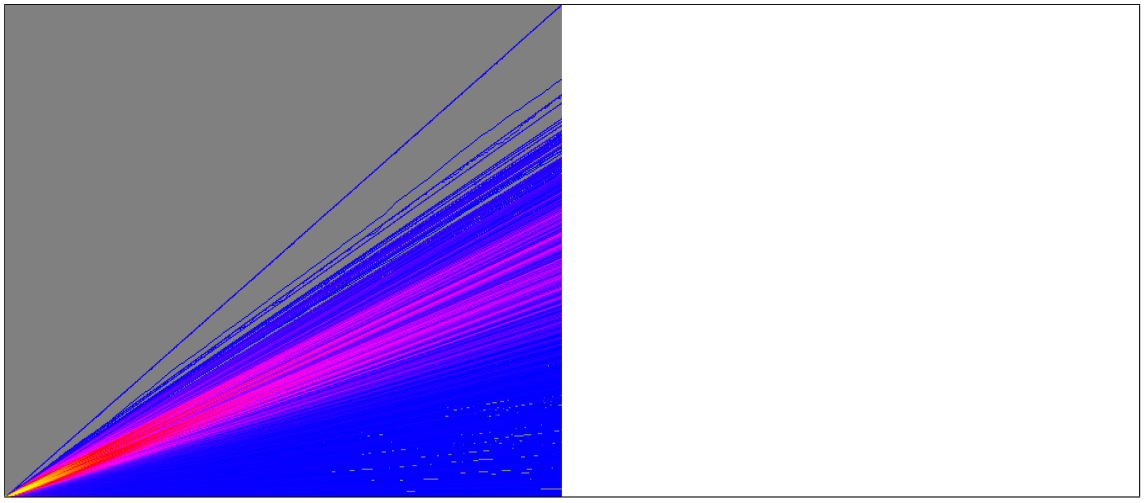


Figure A.8: From Fig. 4.5 with $P = 0$, $C = 0$, $B = 3$

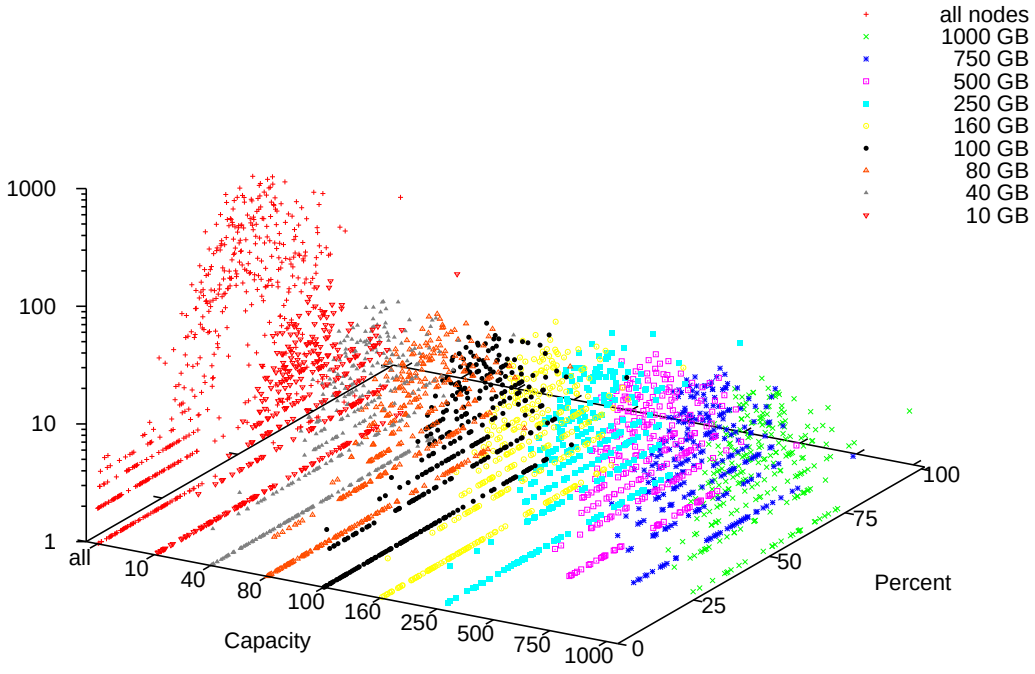


Figure A.9: Node level distribution of Fig. A.8

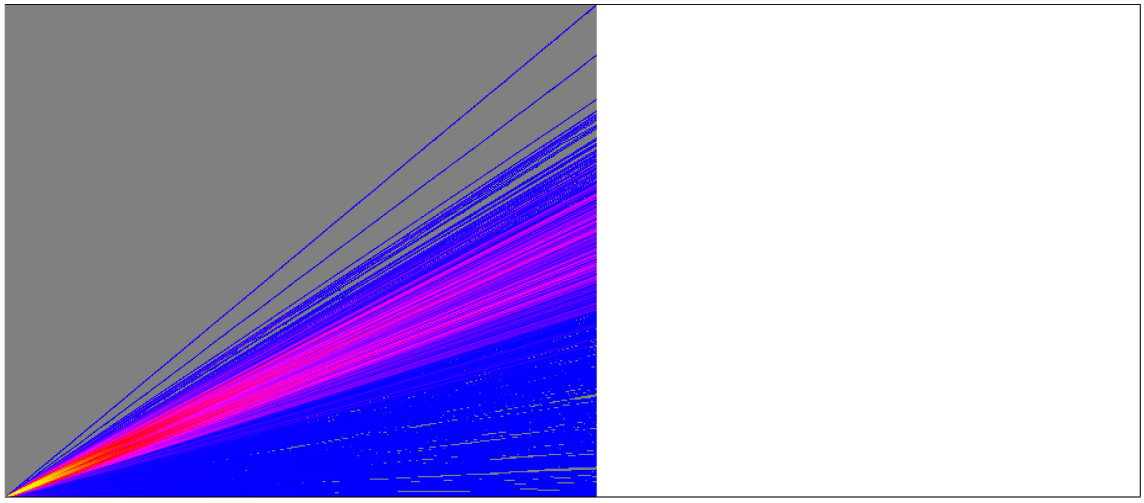


Figure A.10: From Fig. 4.5 with $P = 0$, $C = 0$, $B = 4$

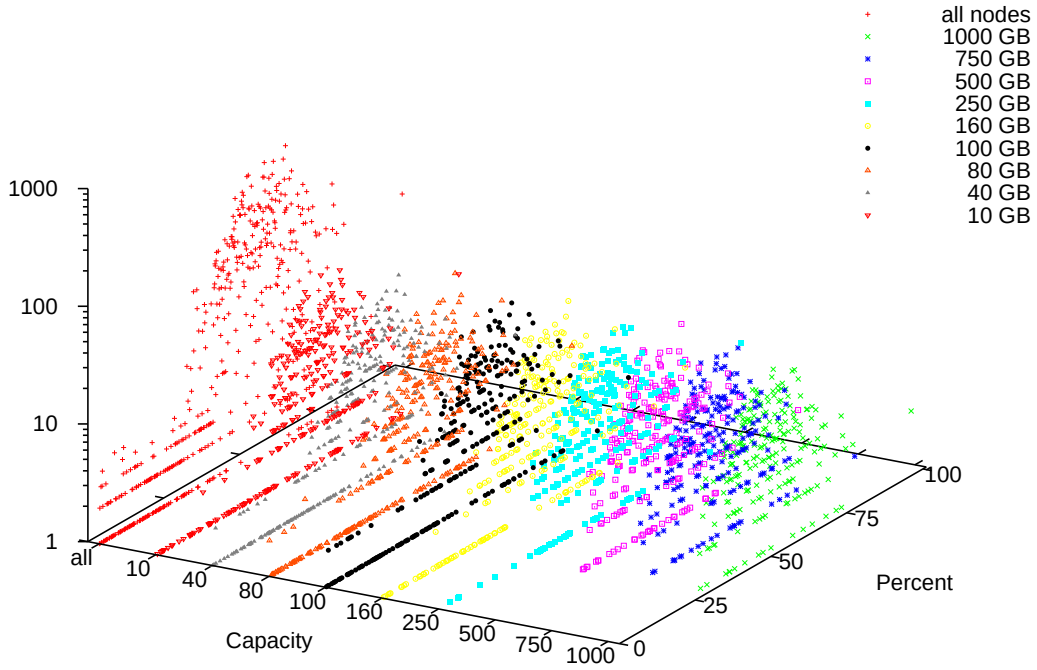


Figure A.11: Node level distribution of Fig. A.10

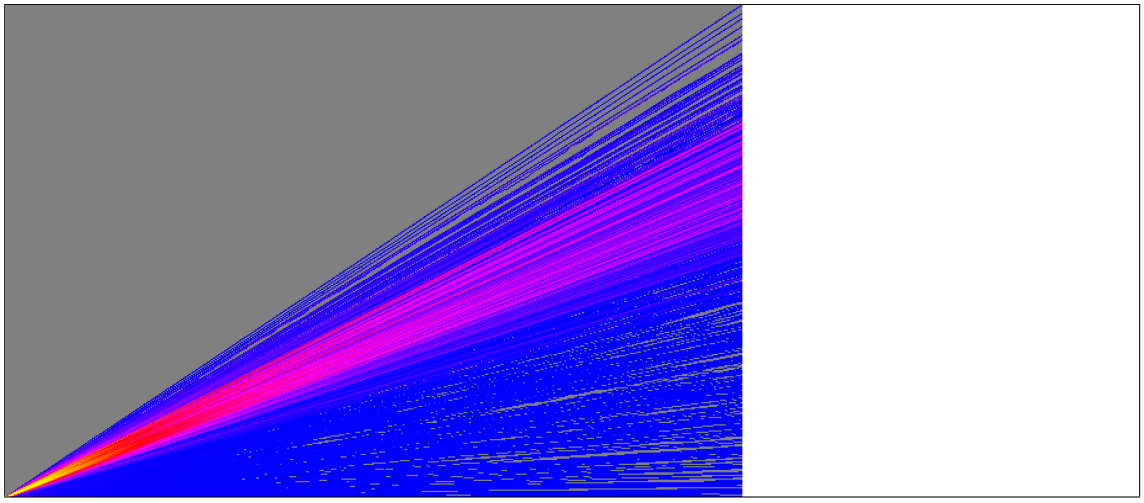


Figure A.12: From Fig. 4.5 with $P = 0$, $C = 0$, $B = 5$

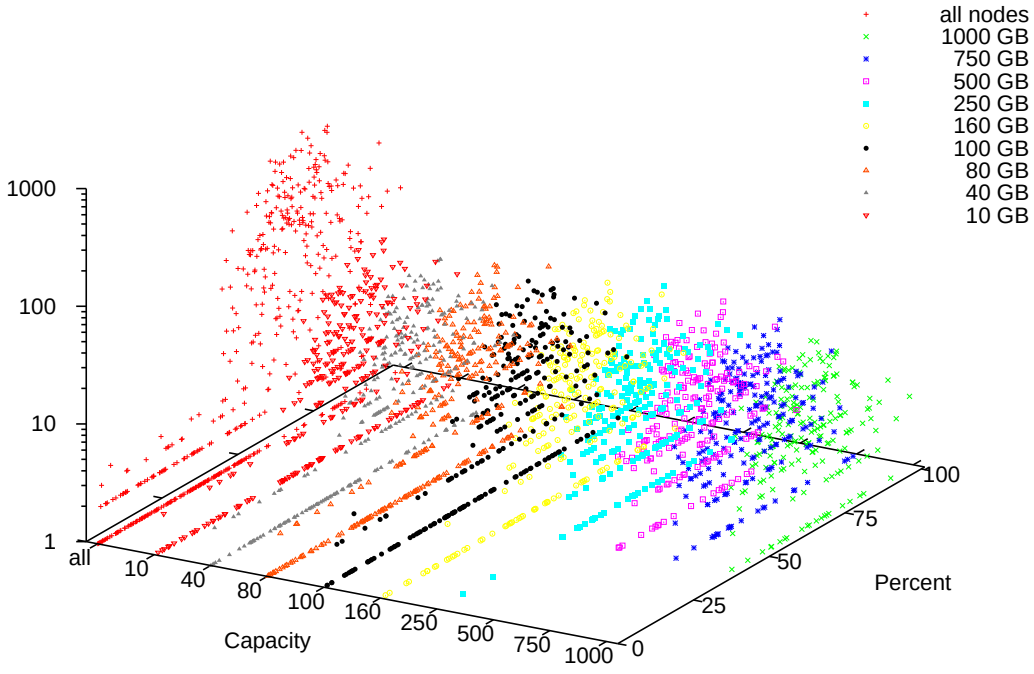


Figure A.13: Node level distribution of Fig. A.12

Figure A.14: From Fig. 4.5 with $P = 0$, $C = 1$, $B = 0$

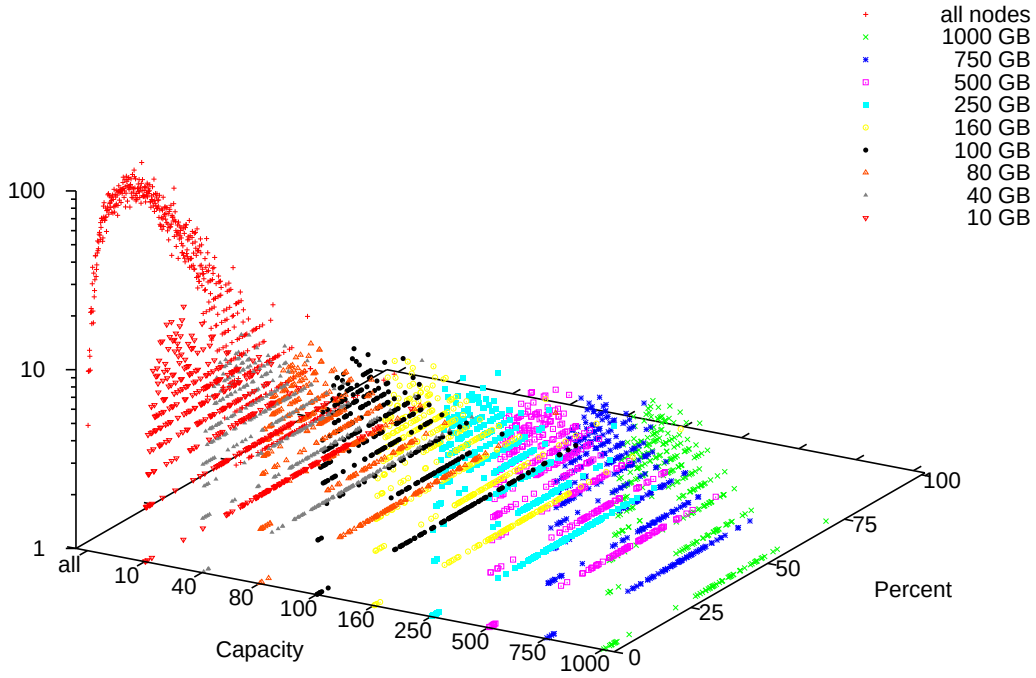


Figure A.15: Node level distribution of Fig. A.14

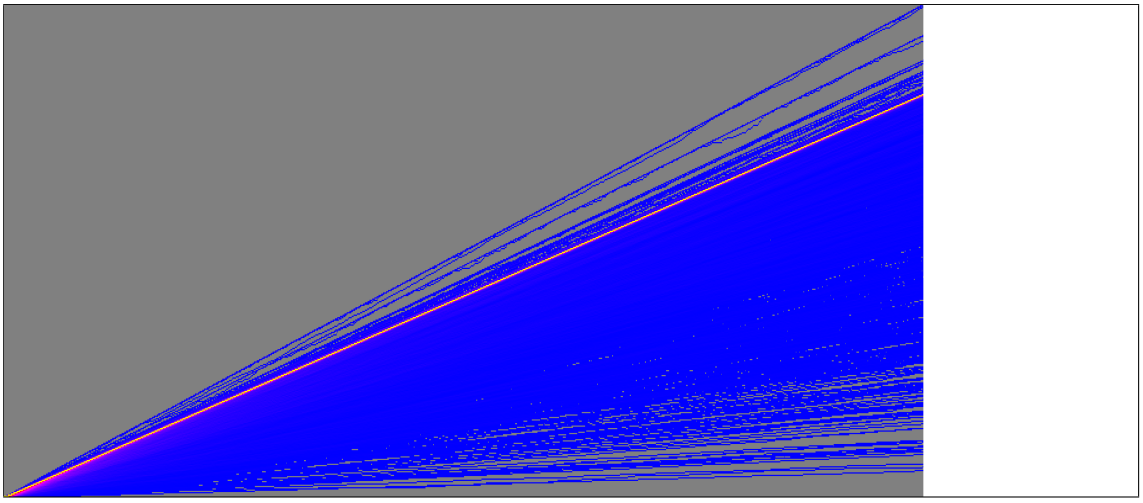


Figure A.16: From Fig. 4.5 with $P = 0$, $C = 1$, $B = 1$

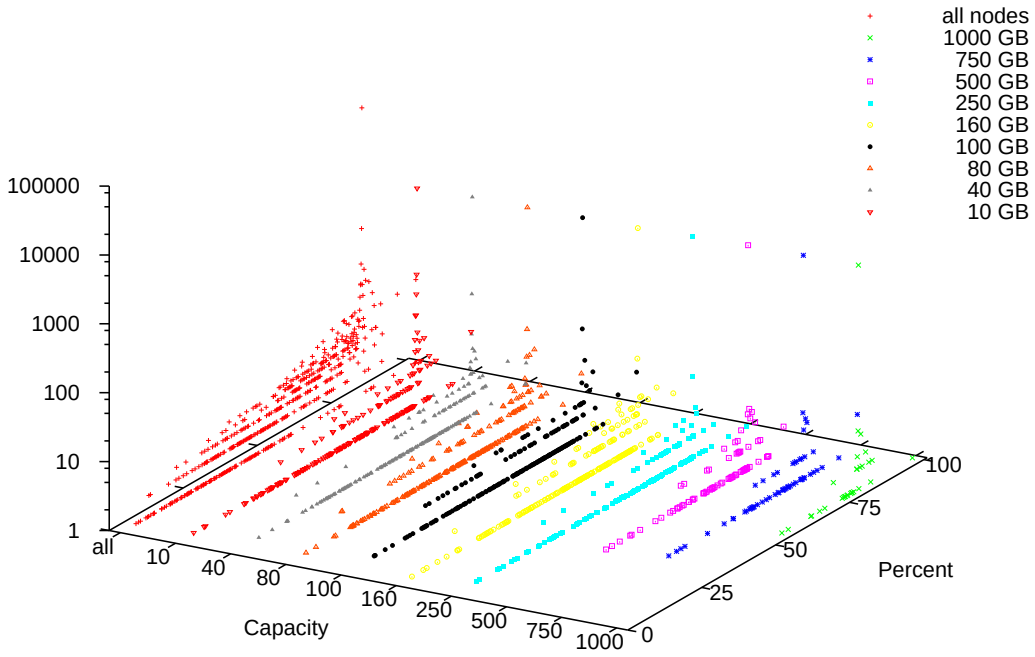


Figure A.17: Node level distribution of Fig. A.16

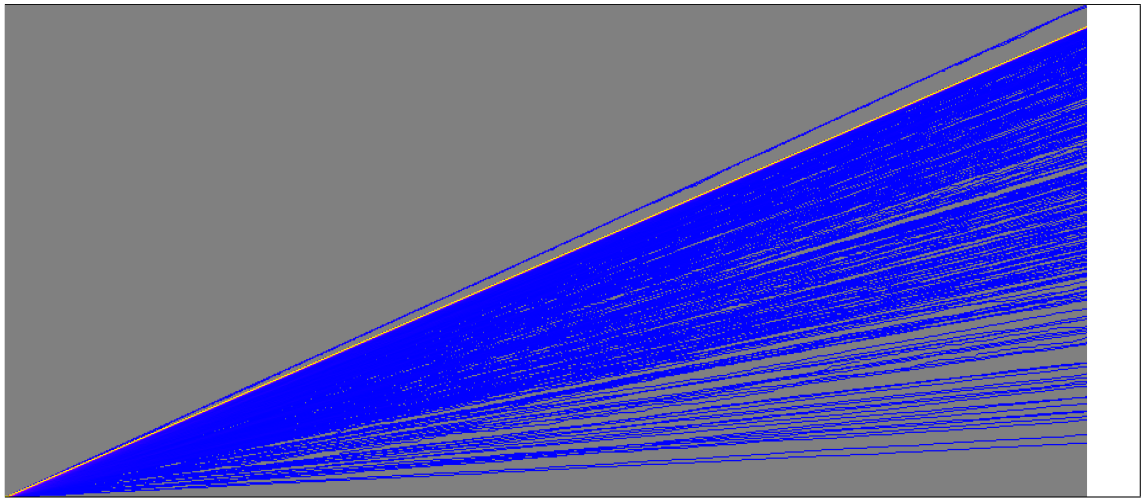


Figure A.18: From Fig. 4.5 with $P = 0$, $C = 1$, $B = 2$

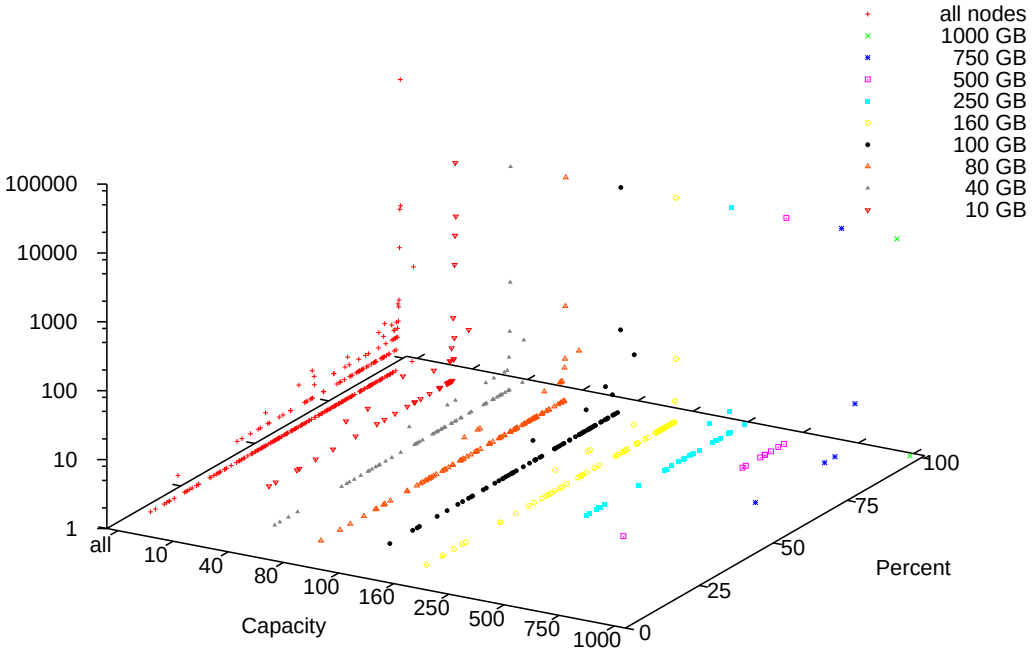


Figure A.19: Node level distribution of Fig. A.18

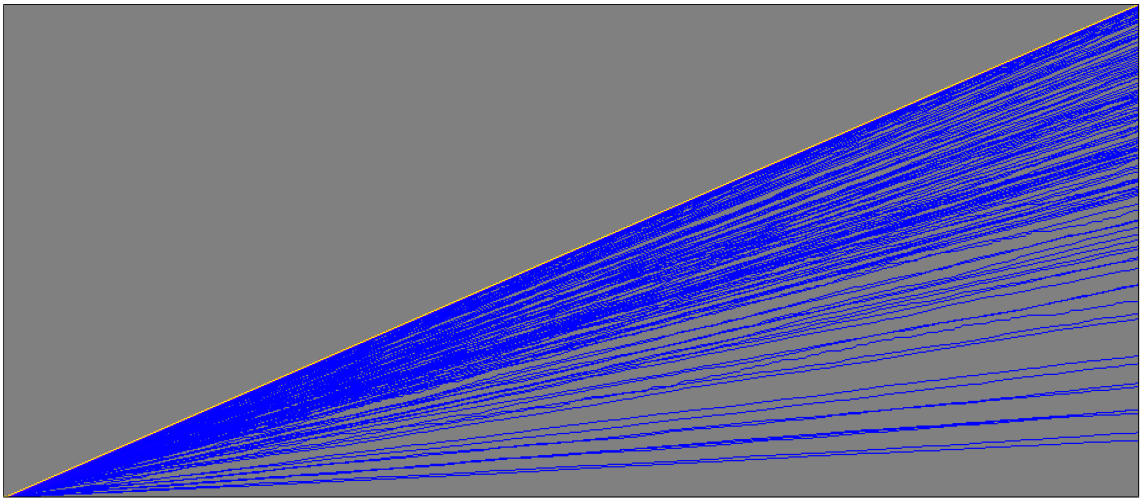


Figure A.20: From Fig. 4.5 with $P = 0$, $C = 1$, $B = 3$

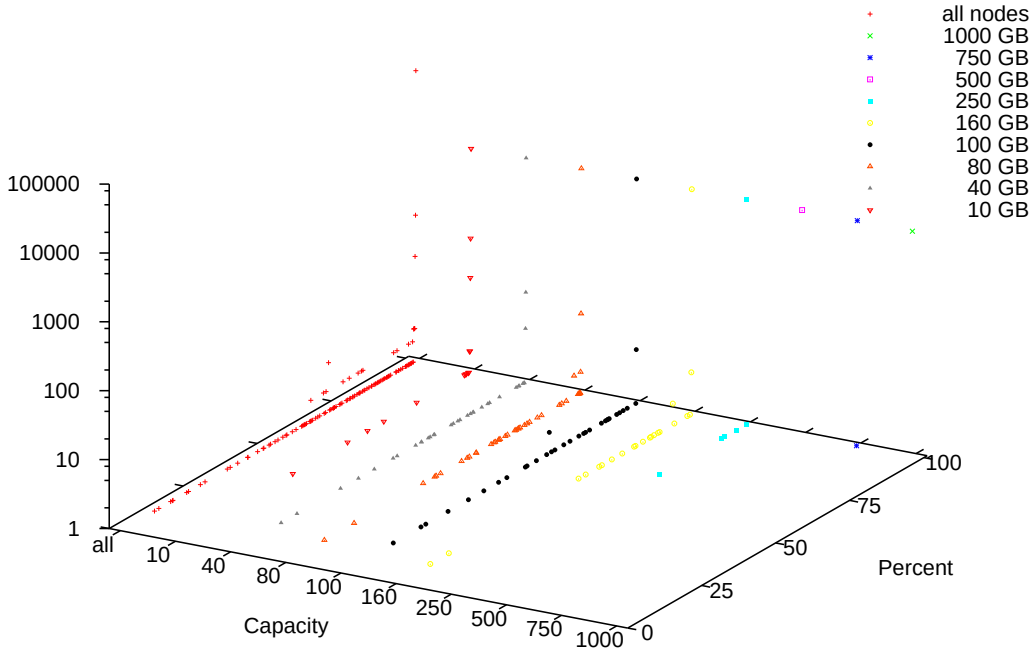


Figure A.21: Node level distribution of Fig. A.20

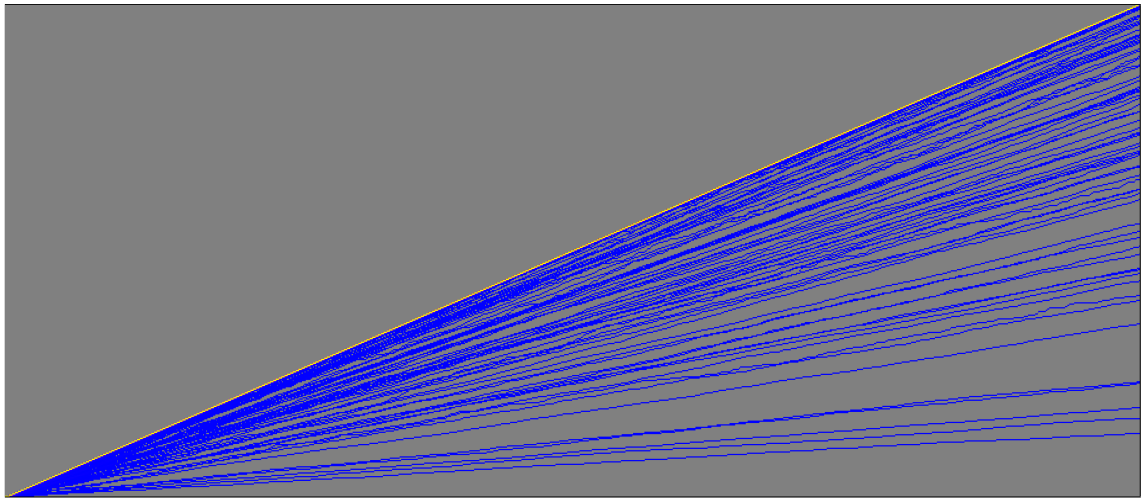


Figure A.22: From Fig. 4.5 with $P = 0$, $C = 1$, $B = 4$

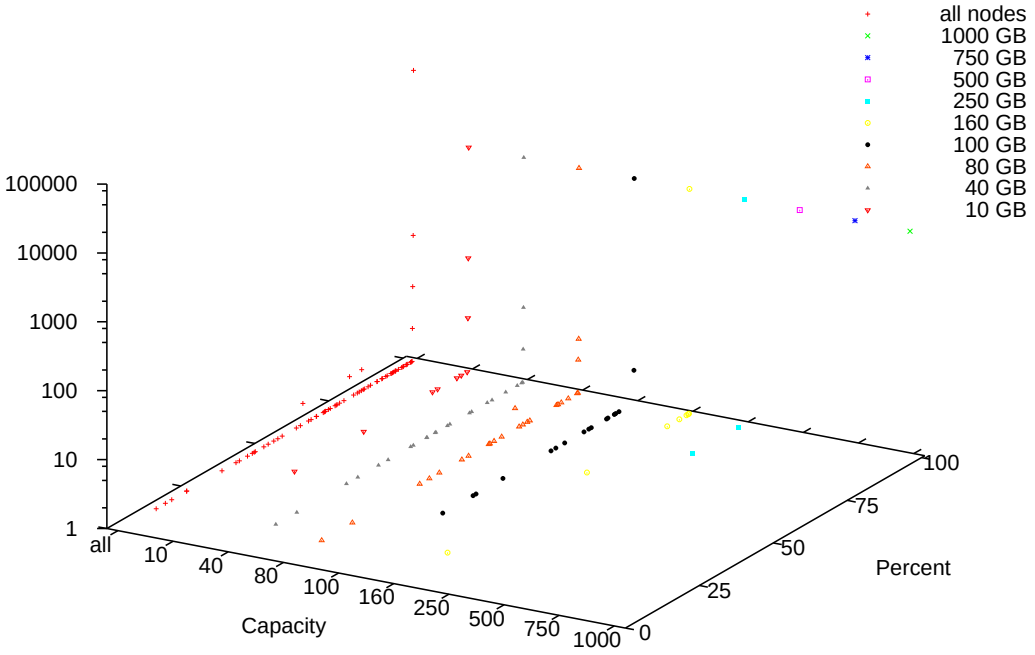


Figure A.23: Node level distribution of Fig. A.22

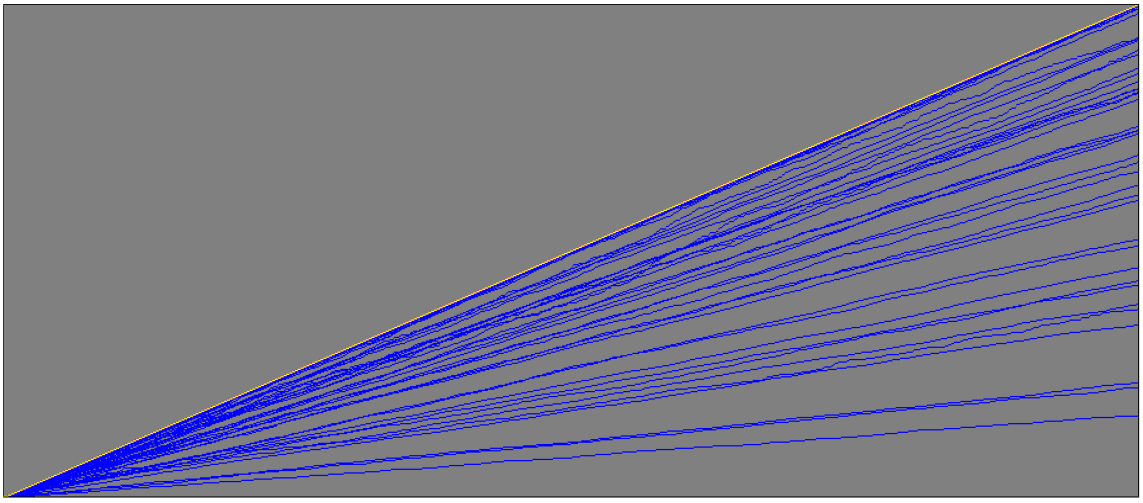


Figure A.24: From Fig. 4.5 with $P = 0$, $C = 1$, $B = 5$

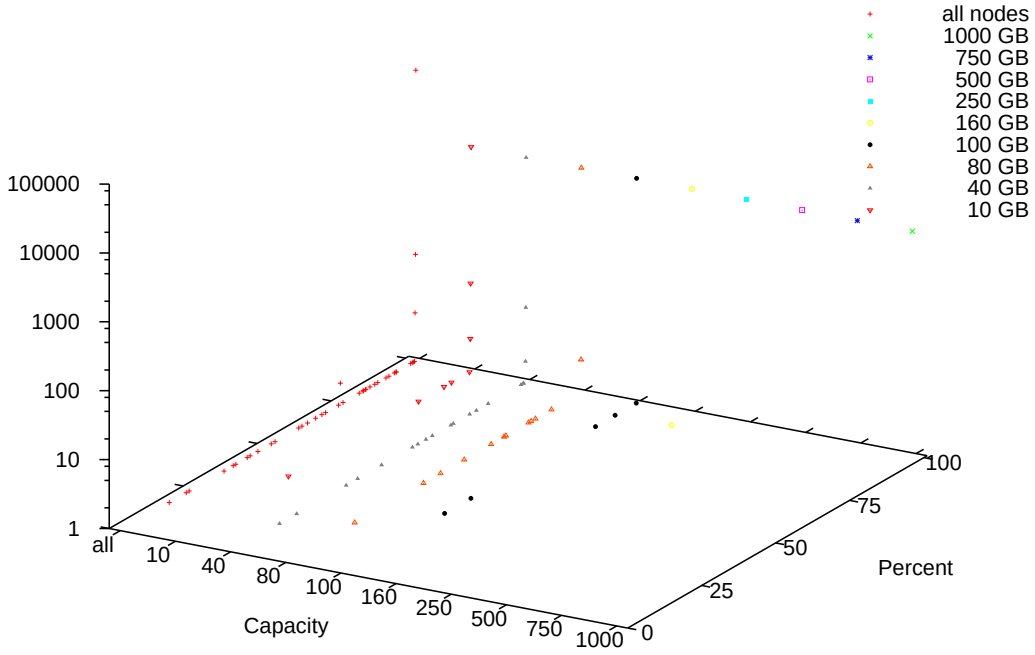


Figure A.25: Node level distribution of Fig. A.24

Figure A.26: From Fig. 4.5 with $P = 0$, $C = 2$, $B = 0$

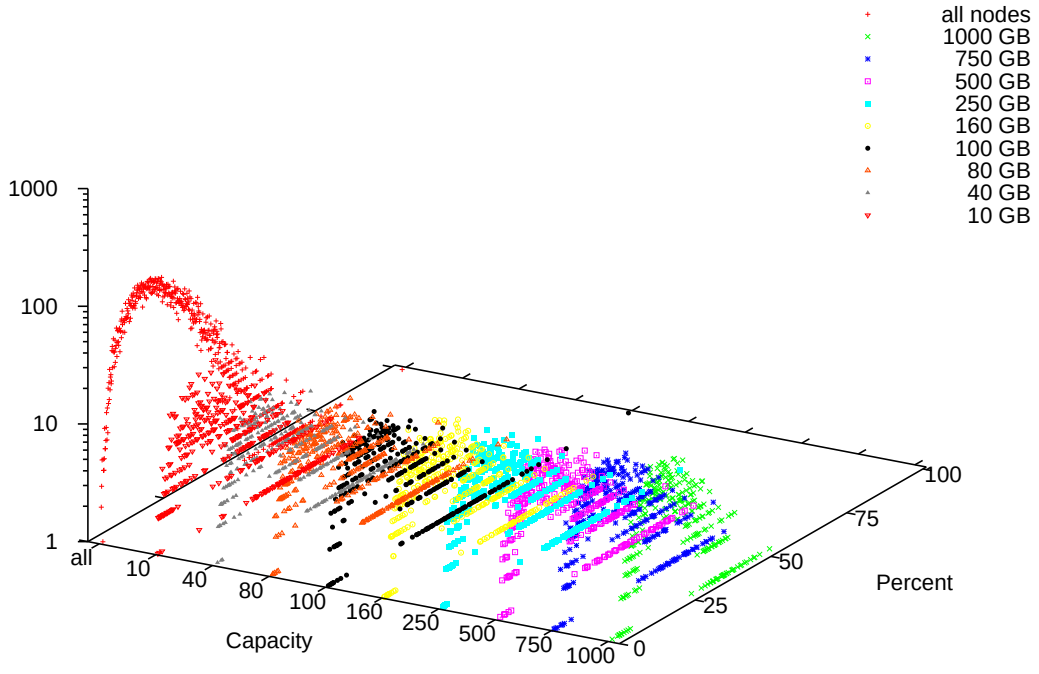


Figure A.27: Node level distribution of Fig. A.26

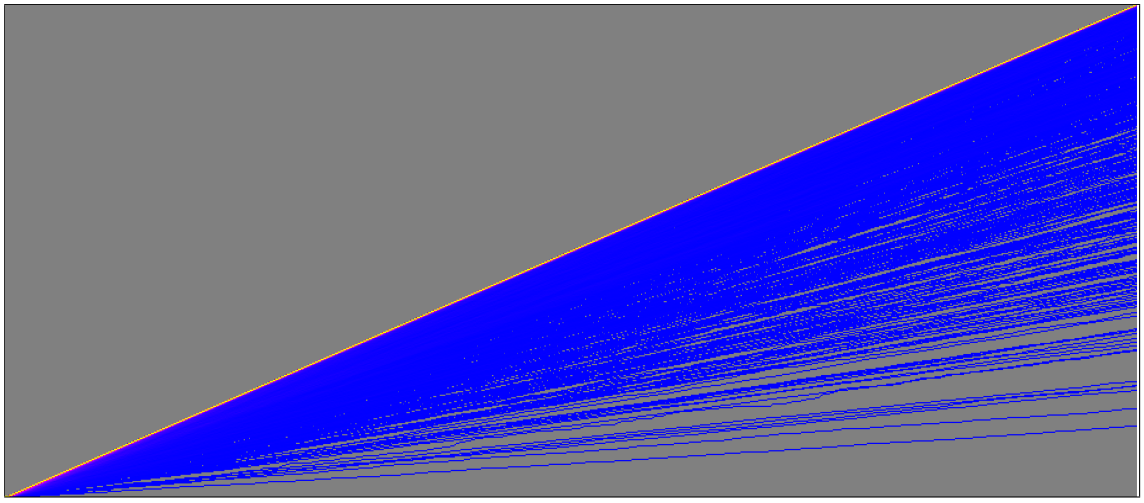


Figure A.28: From Fig. 4.5 with $P = 0$, $C = 2$, $B = 1$

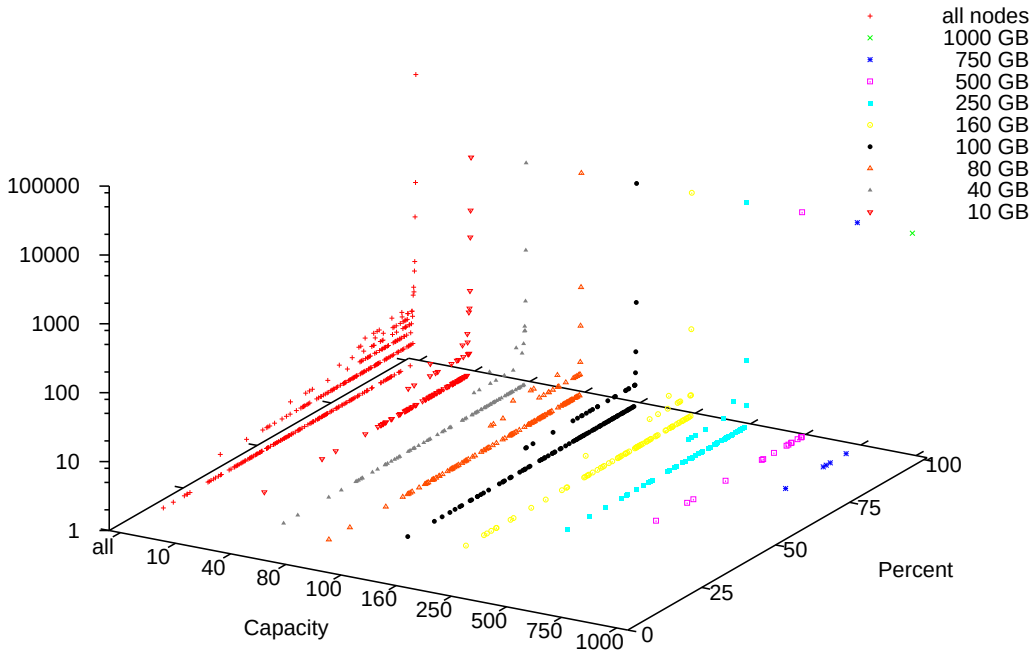


Figure A.29: Node level distribution of Fig. A.28

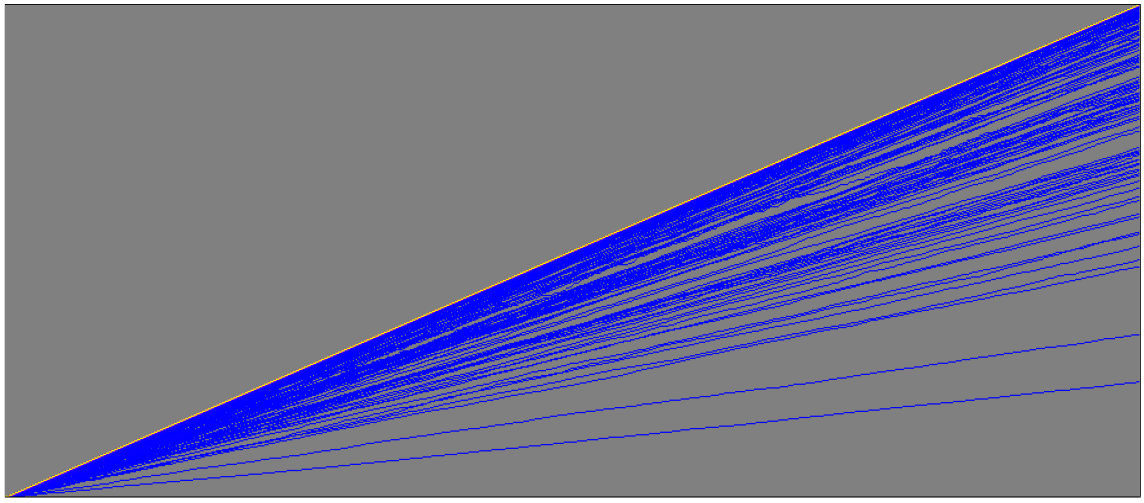


Figure A.30: From Fig. 4.5 with $P = 0$, $C = 2$, $B = 2$

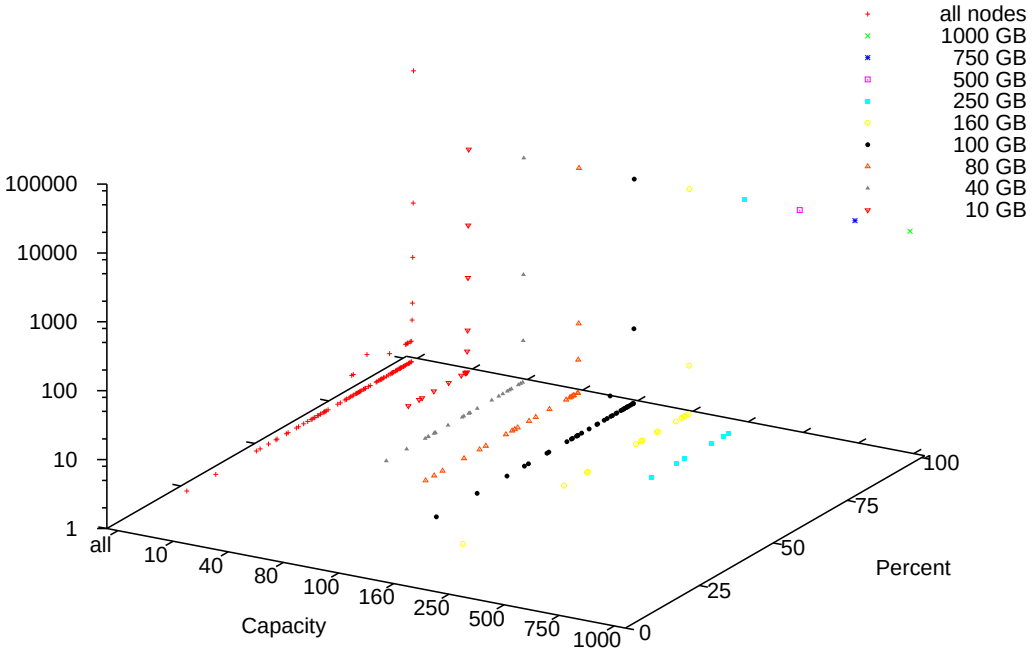


Figure A.31: Node level distribution of Fig. A.30

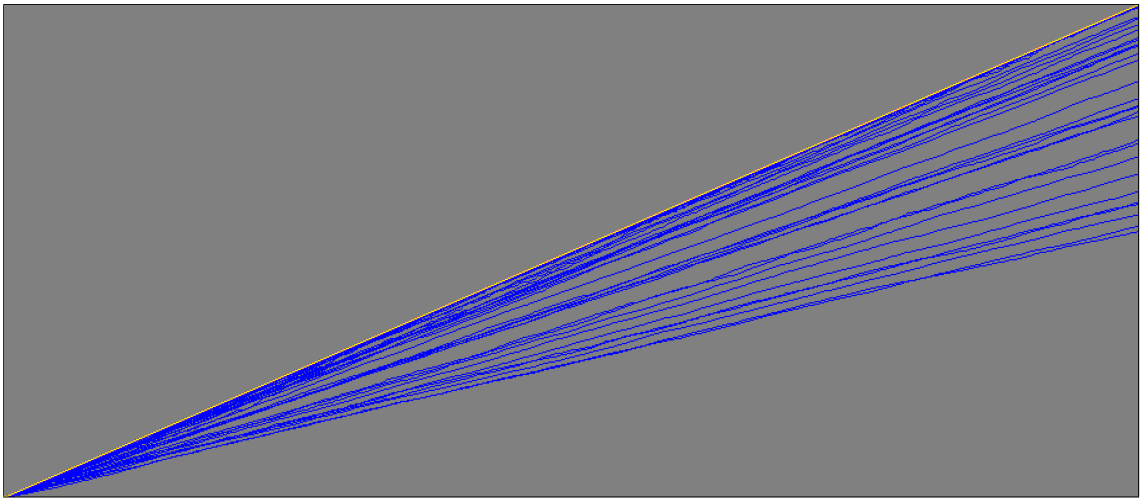


Figure A.32: From Fig. 4.5 with $P = 0$, $C = 2$, $B = 3$

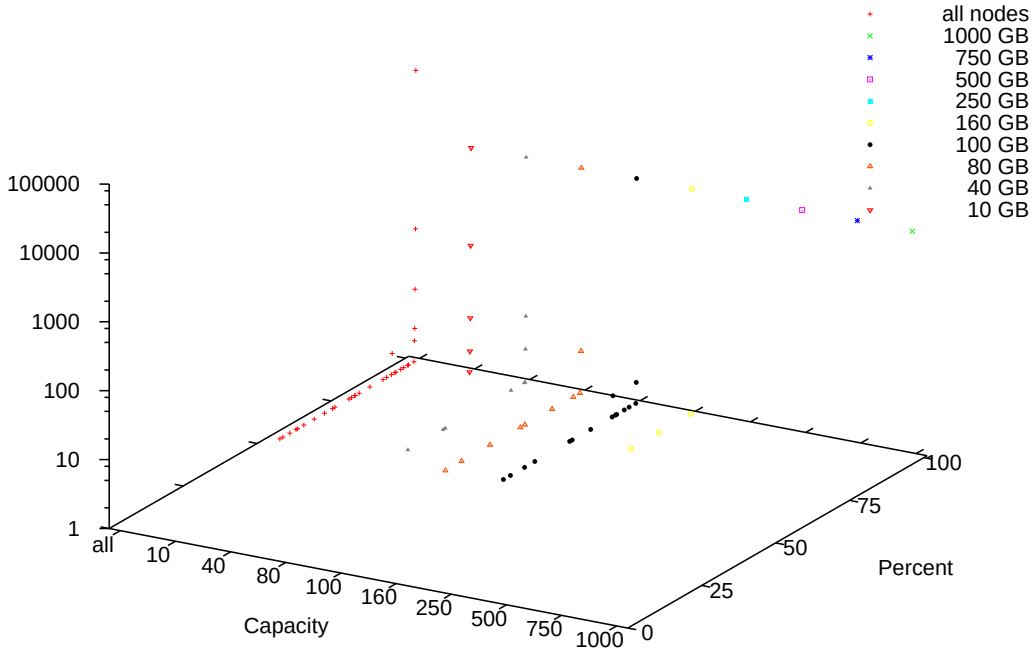


Figure A.33: Node level distribution of Fig. A.32

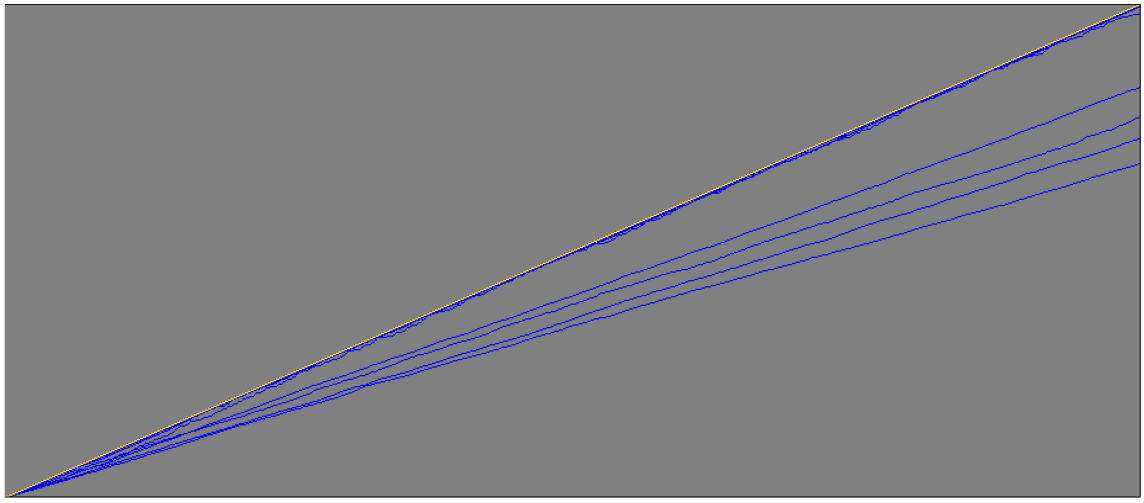


Figure A.34: From Fig. 4.5 with $P = 0$, $C = 2$, $B = 4$

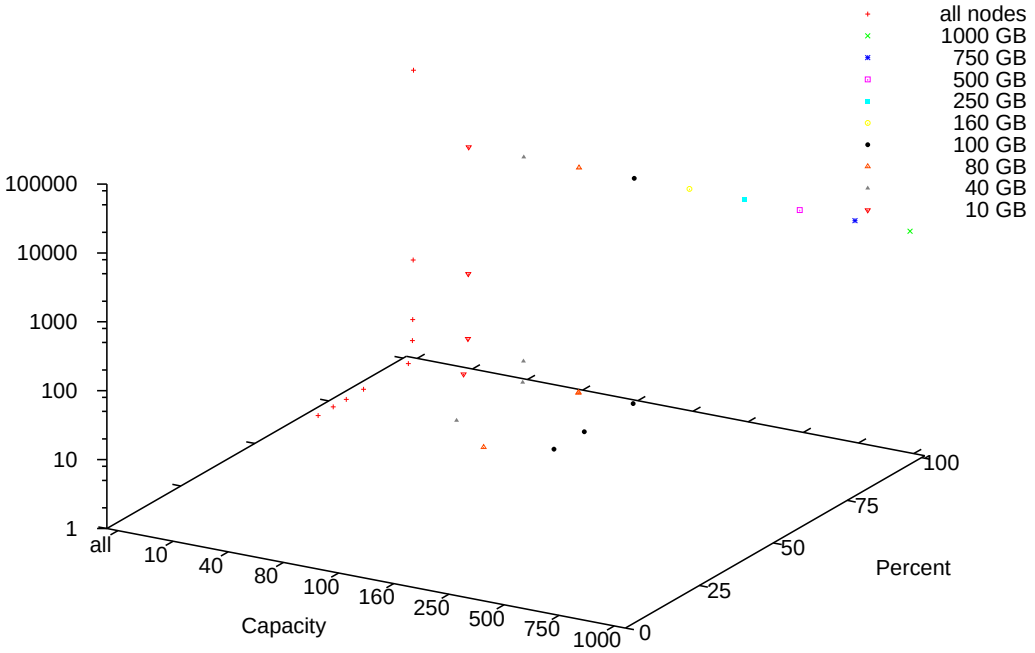


Figure A.35: Node level distribution of Fig. A.34

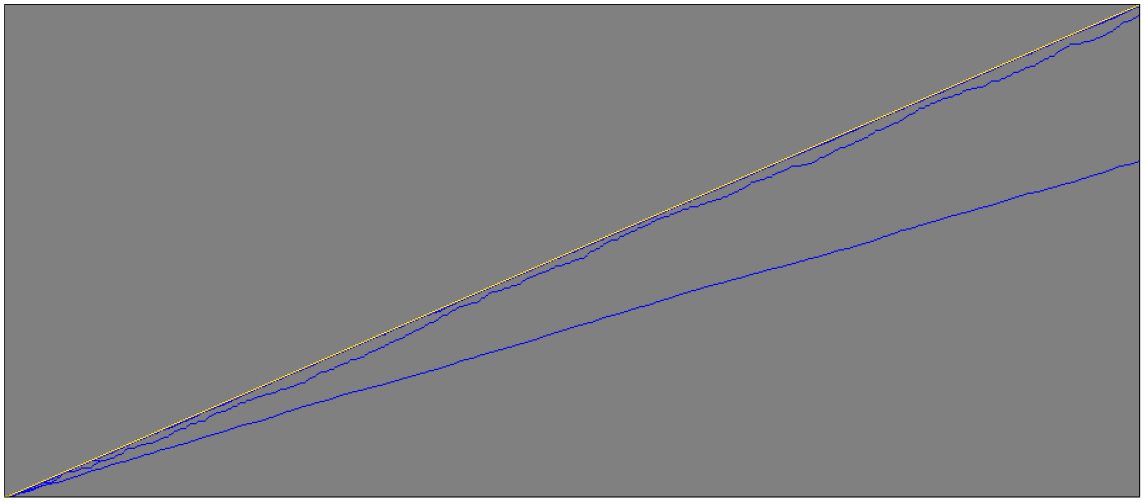


Figure A.36: From Fig. 4.5 with $P = 0$, $C = 2$, $B = 5$

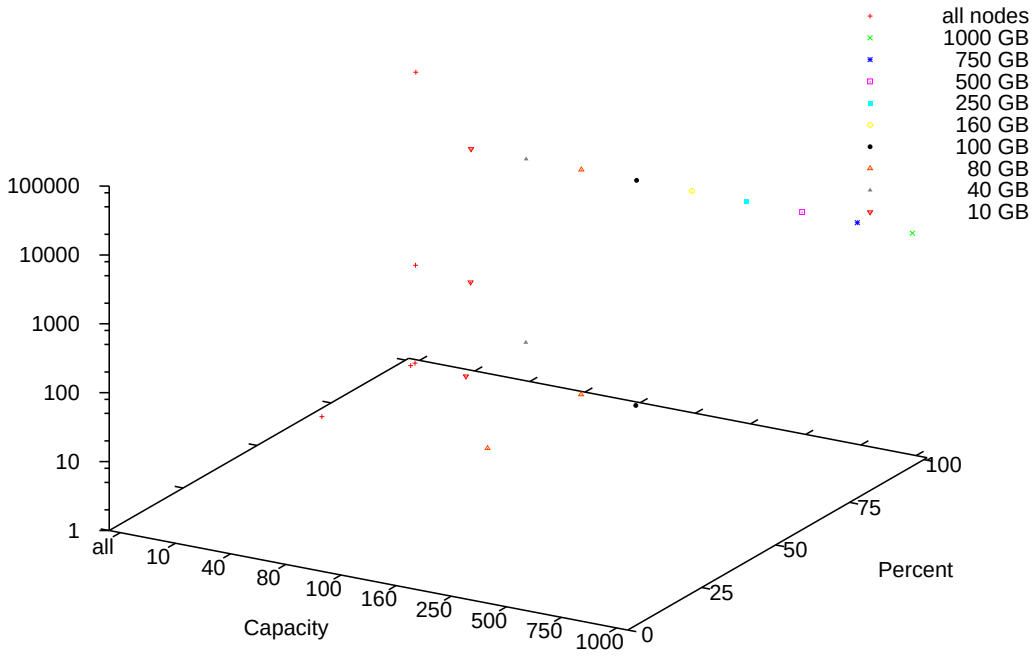


Figure A.37: Node level distribution of Fig. A.36

Figure A.38: From Fig. 4.6 with $P = 0$, $C = 3$, $B = 0$

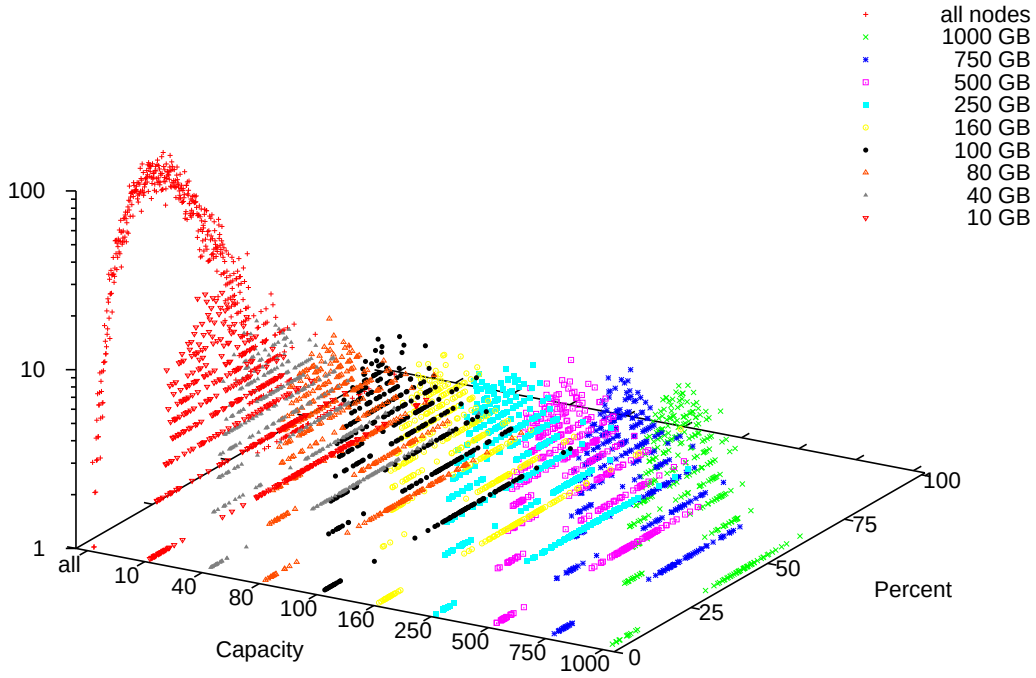


Figure A.39: Node level distribution of Fig. A.38

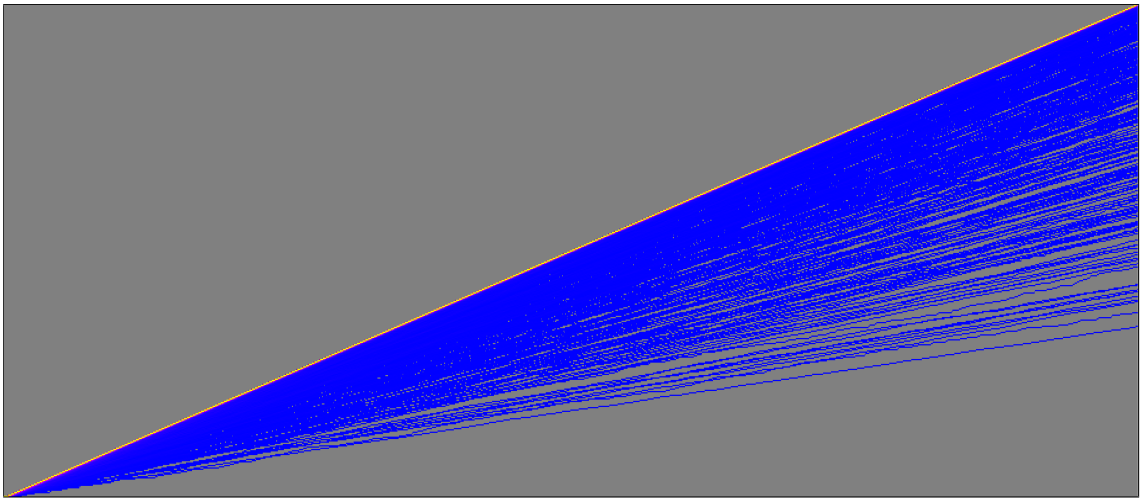


Figure A.40: From Fig. 4.6 with $P = 0$, $C = 3$, $B = 1$

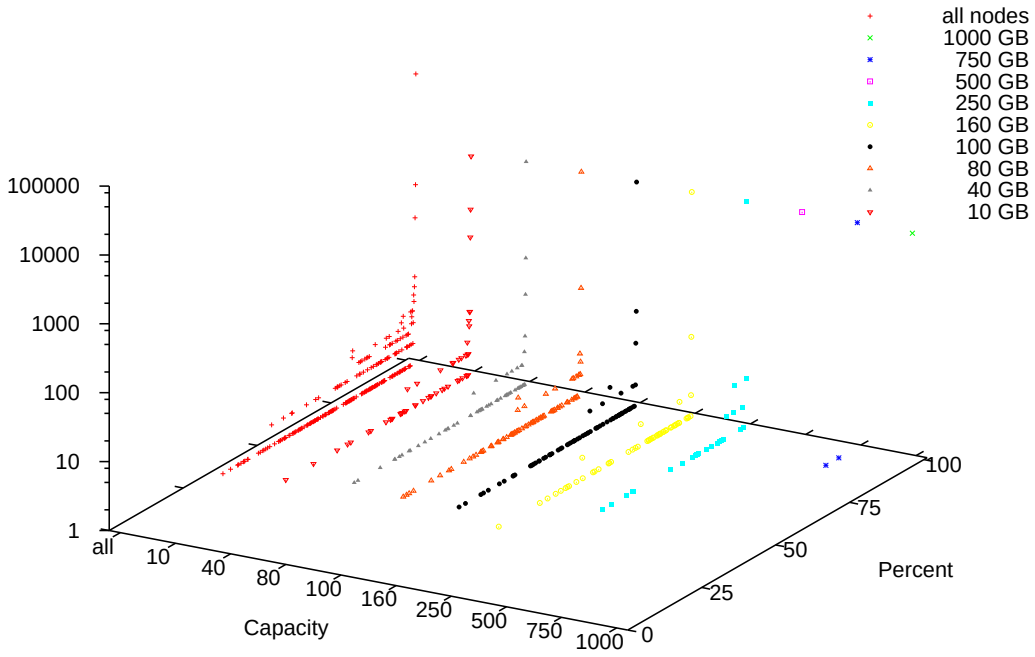


Figure A.41: Node level distribution of Fig. A.40

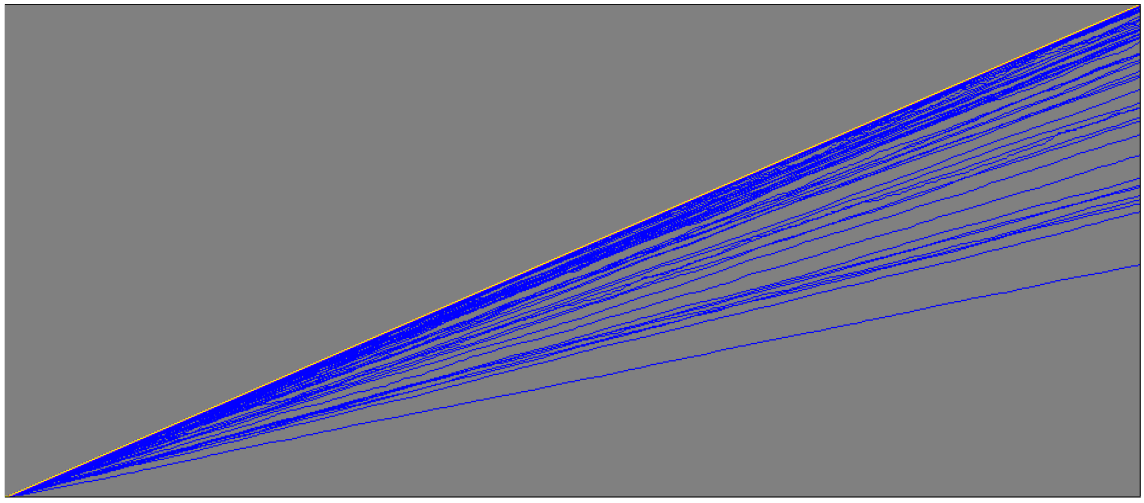


Figure A.42: From Fig. 4.6 with $P = 0$, $C = 3$, $B = 2$

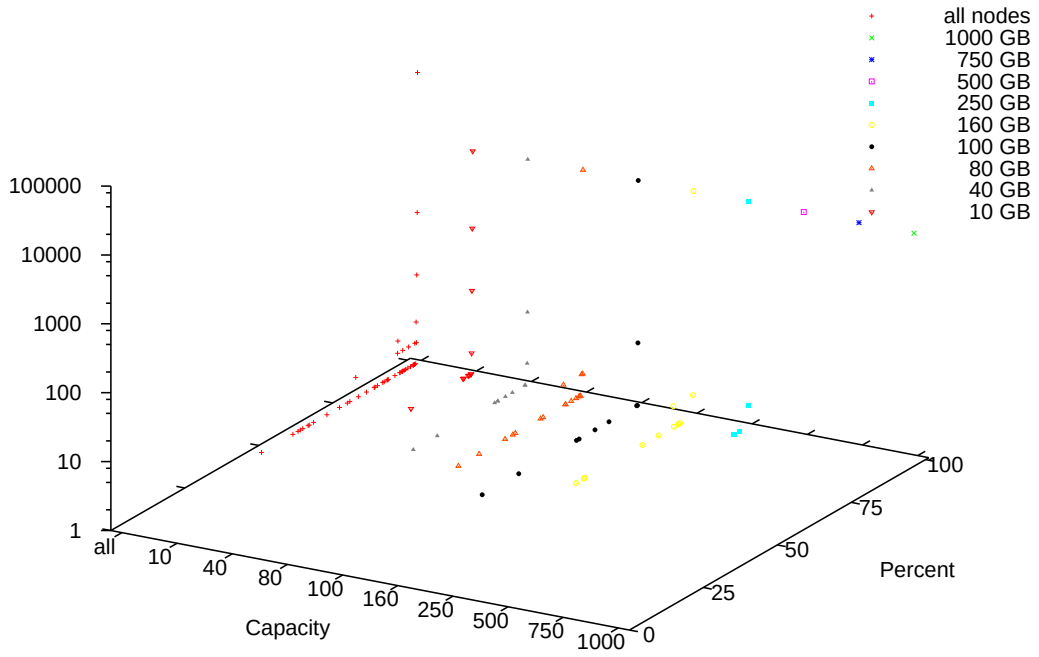


Figure A.43: Node level distribution of Fig. A.42

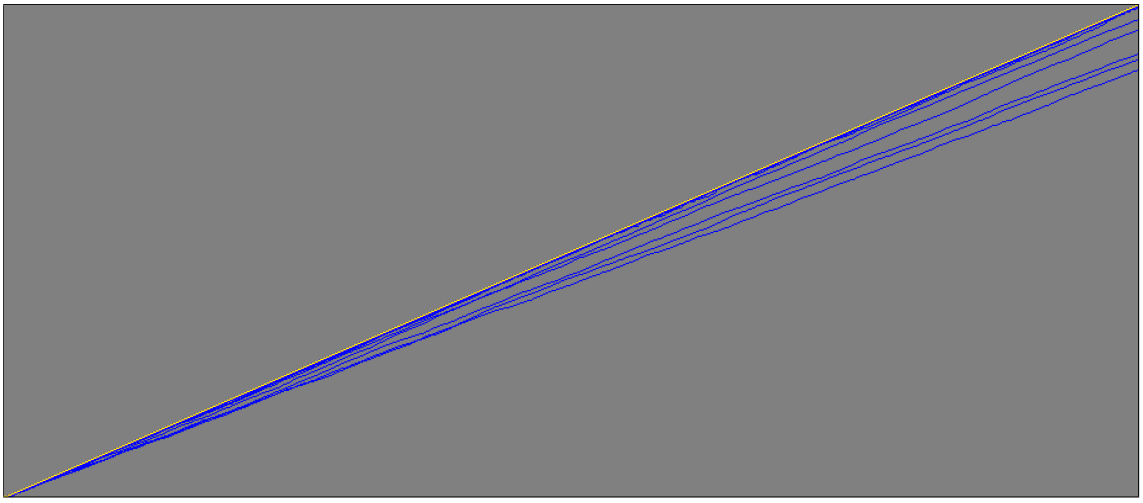


Figure A.44: From Fig. 4.6 with $P = 0$, $C = 3$, $B = 3$

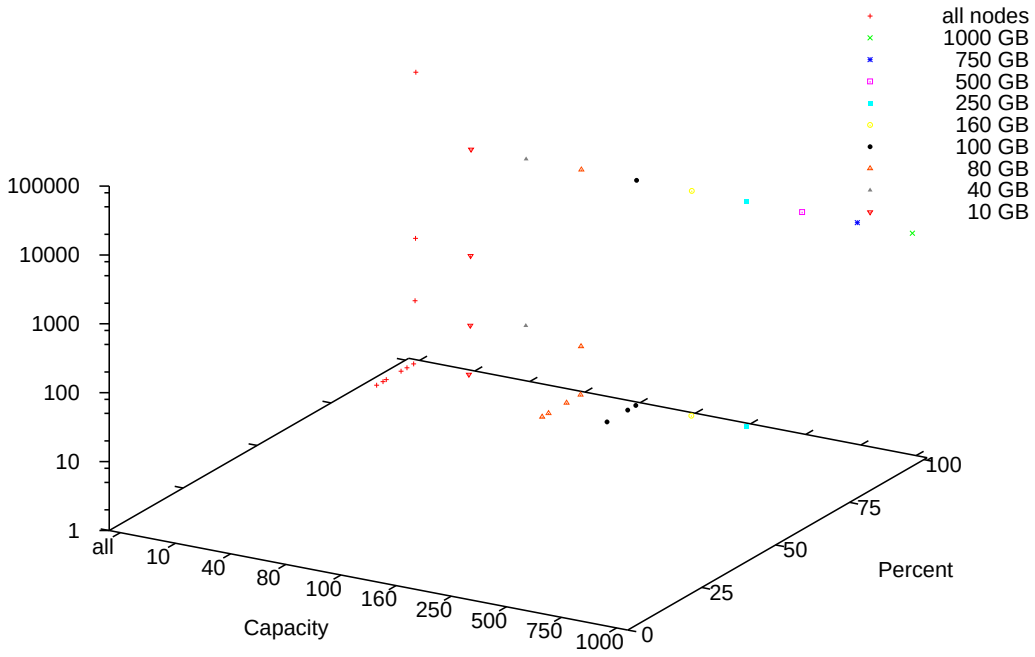


Figure A.45: Node level distribution of Fig. A.44

Figure A.46: From Fig. 4.6 with $P = 0$, $C = 4$, $B = 0$

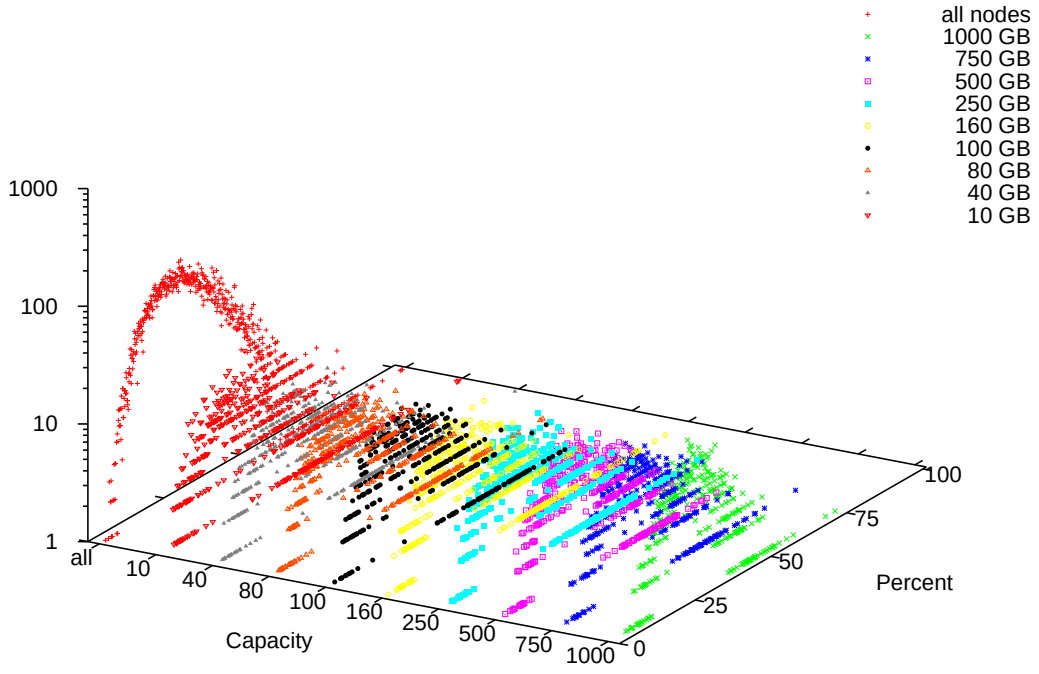


Figure A.47: Node level distribution of Fig. A.46

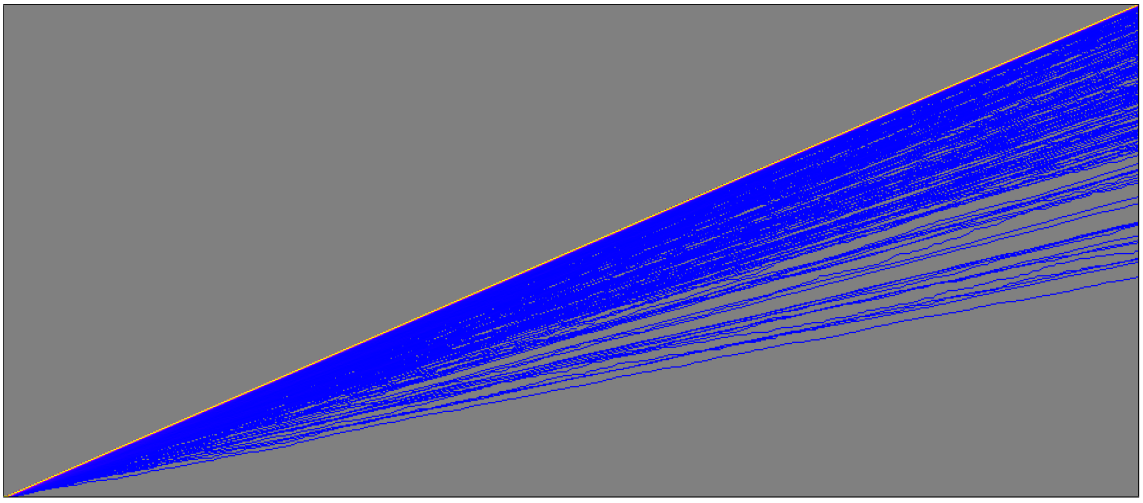


Figure A.48: From Fig. 4.6 with $P = 0$, $C = 4$, $B = 1$

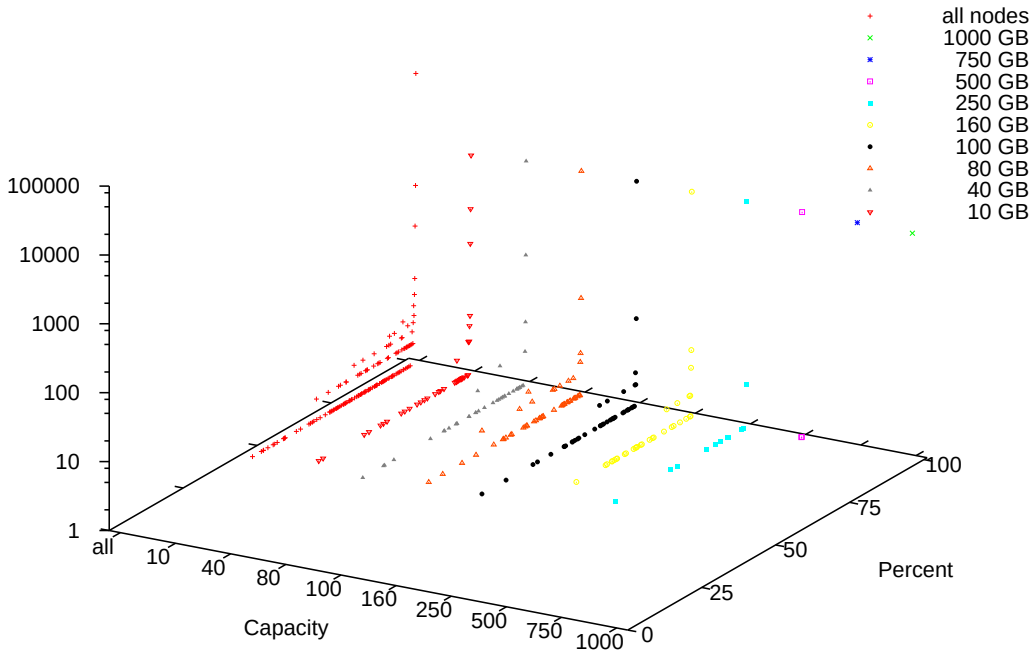


Figure A.49: Node level distribution of Fig. A.48

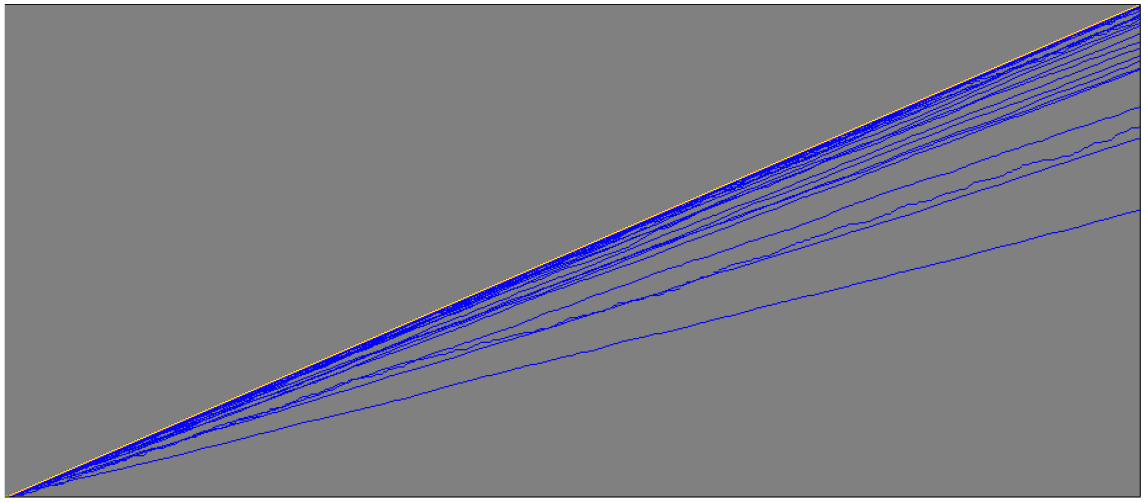


Figure A.50: From Fig. 4.6 with $P = 0$, $C = 4$, $B = 2$

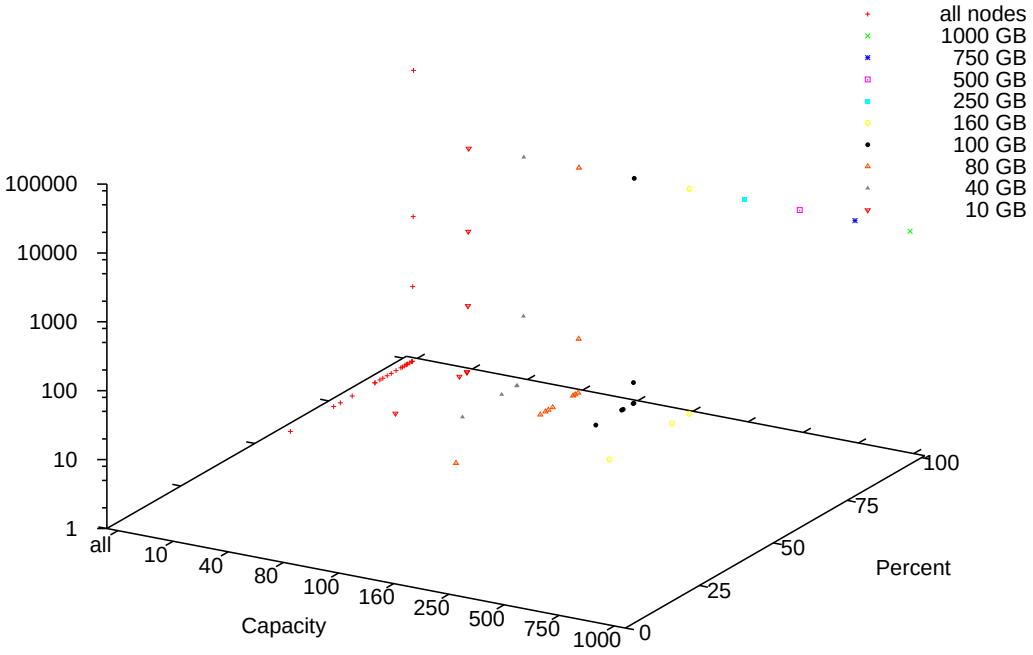


Figure A.51: Node level distribution of Fig. A.50

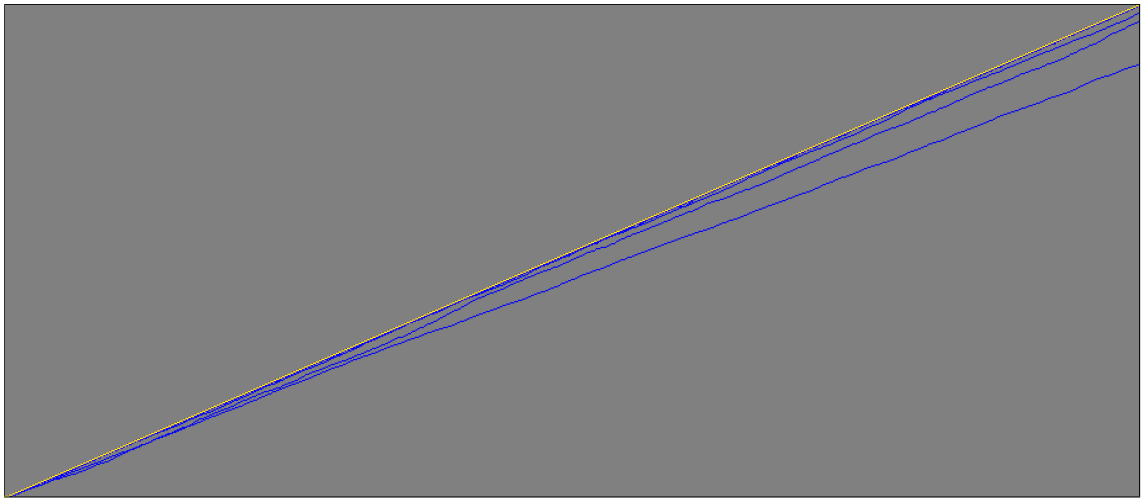


Figure A.52: From Fig. 4.6 with $P = 0$, $C = 4$, $B = 3$

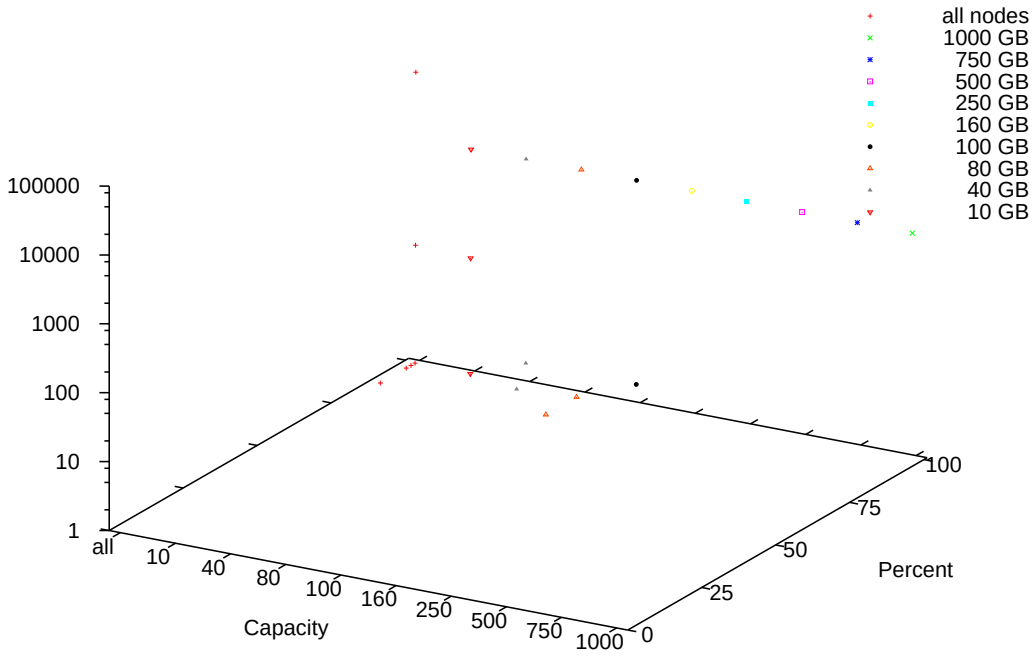


Figure A.53: Node level distribution of Fig. A.52

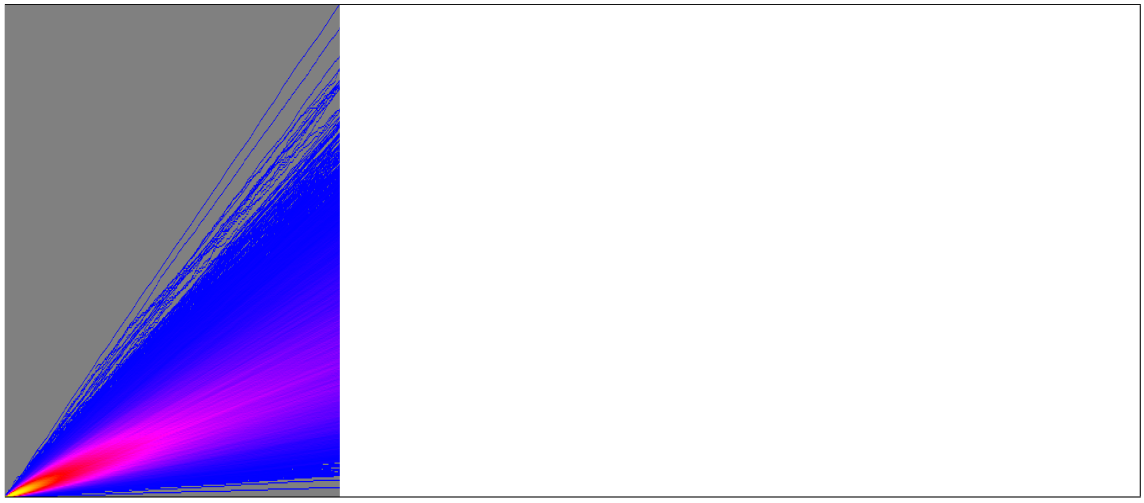


Figure A.54: From Fig. 4.6 with $P = 0$, $C = 5$, $B = 0$

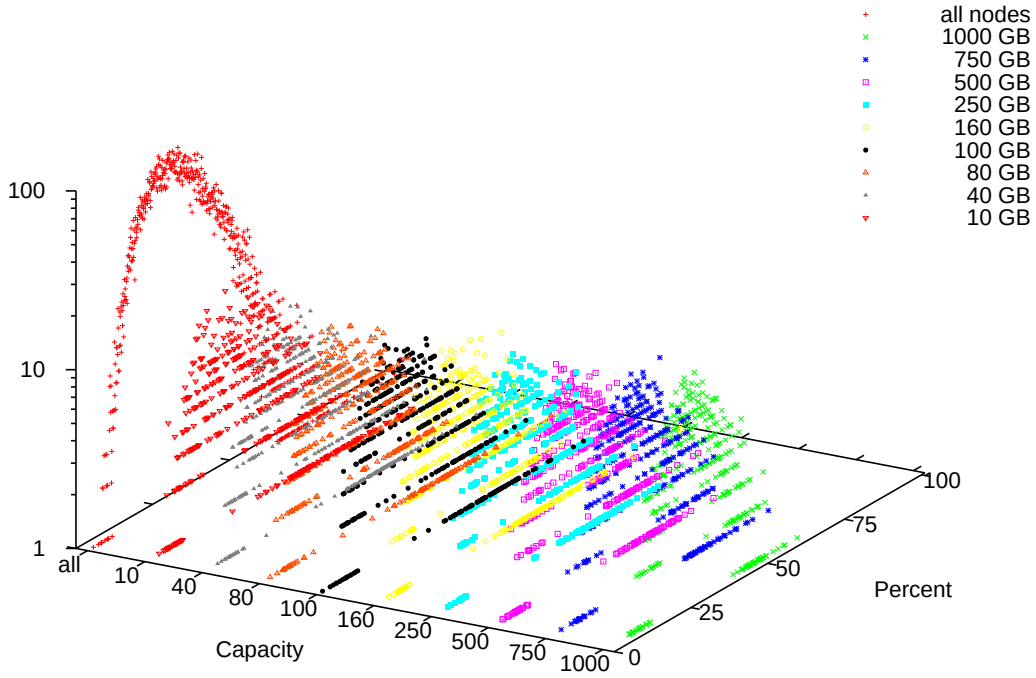


Figure A.55: Node level distribution of Fig. A.54

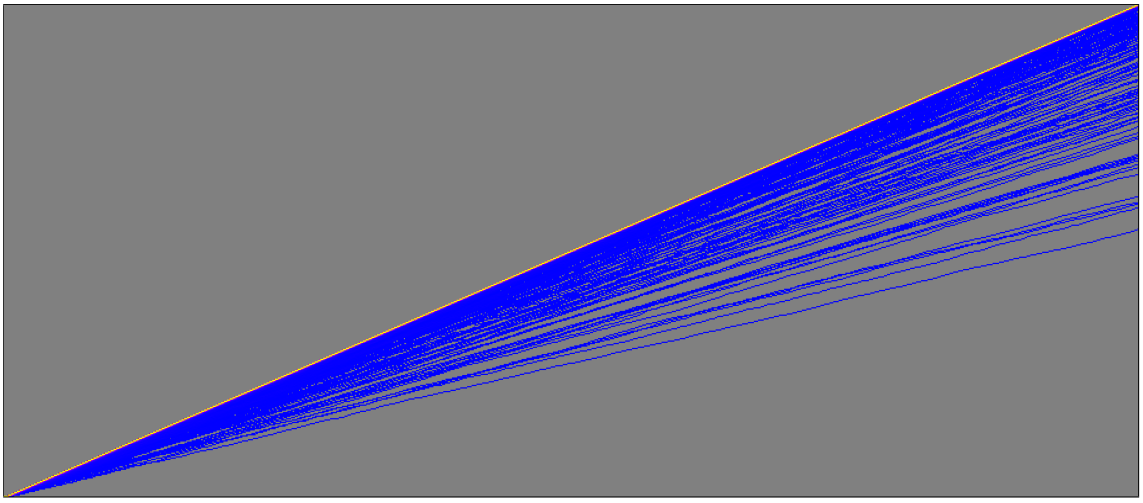


Figure A.56: From Fig. 4.6 with $P = 0$, $C = 5$, $B = 1$

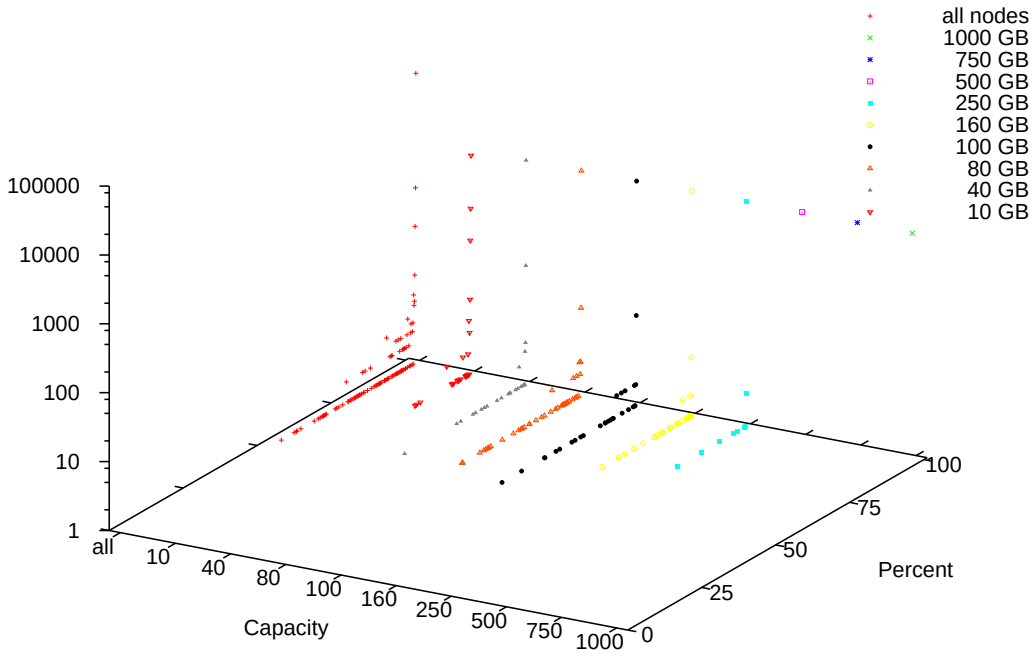


Figure A.57: Node level distribution of Fig. A.56

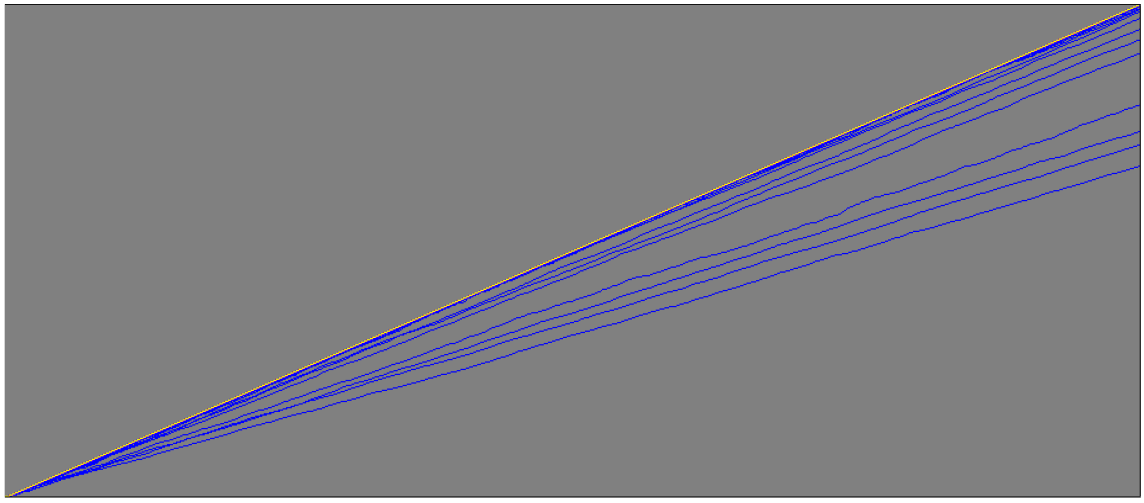


Figure A.58: From Fig. 4.6 with $P = 0$, $C = 5$, $B = 2$

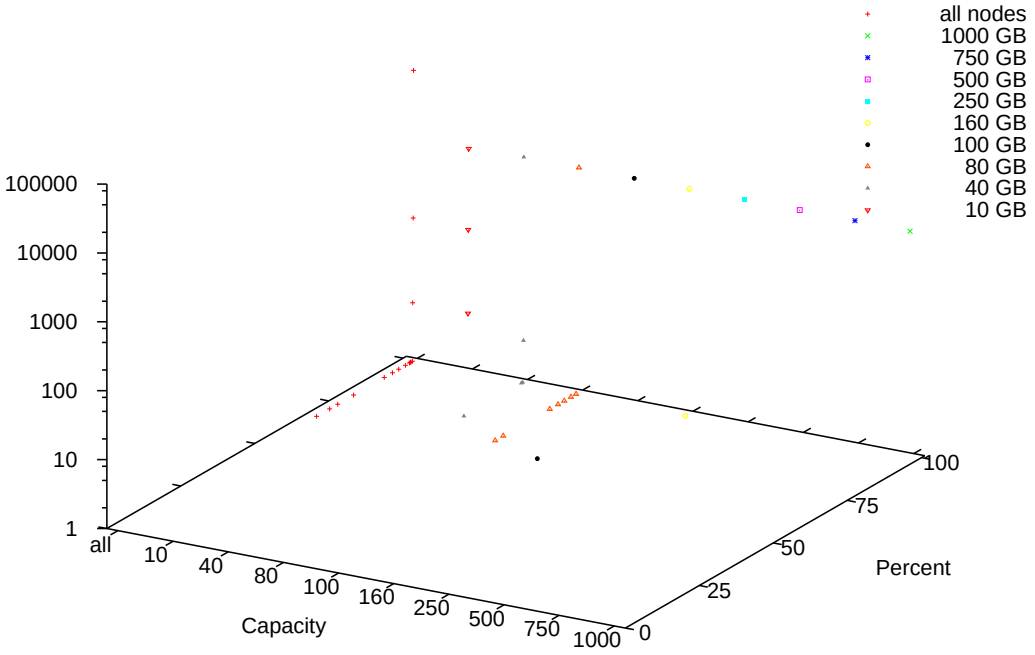


Figure A.59: Node level distribution of Fig. A.58

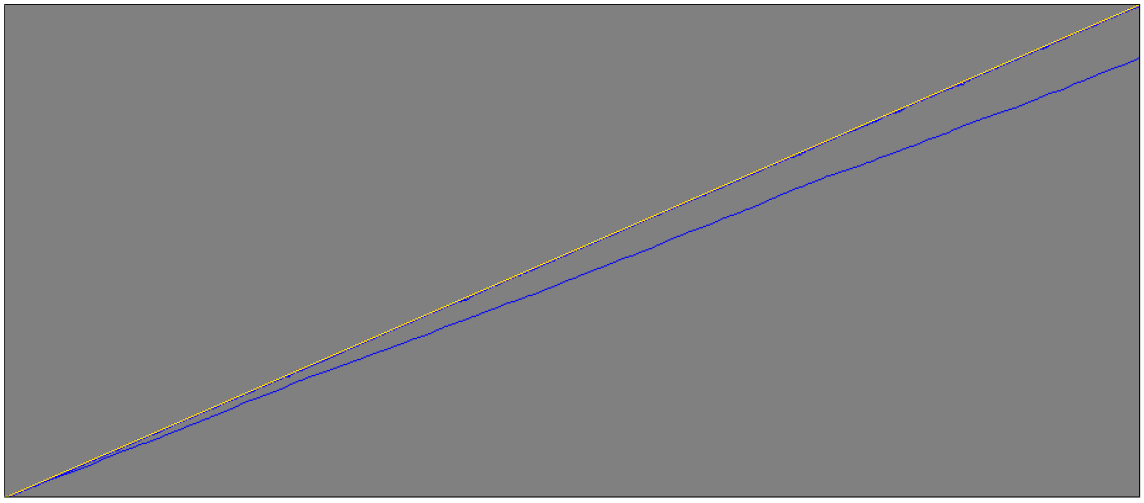


Figure A.60: From Fig. 4.6 with $P = 0$, $C = 5$, $B = 3$

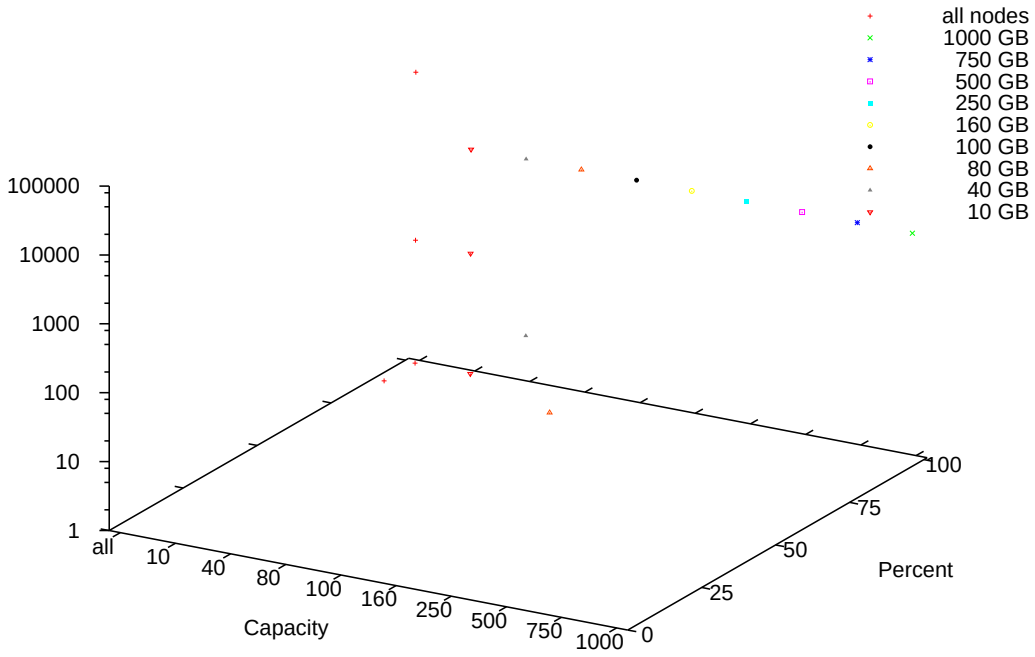


Figure A.61: Node level distribution of Fig. A.60

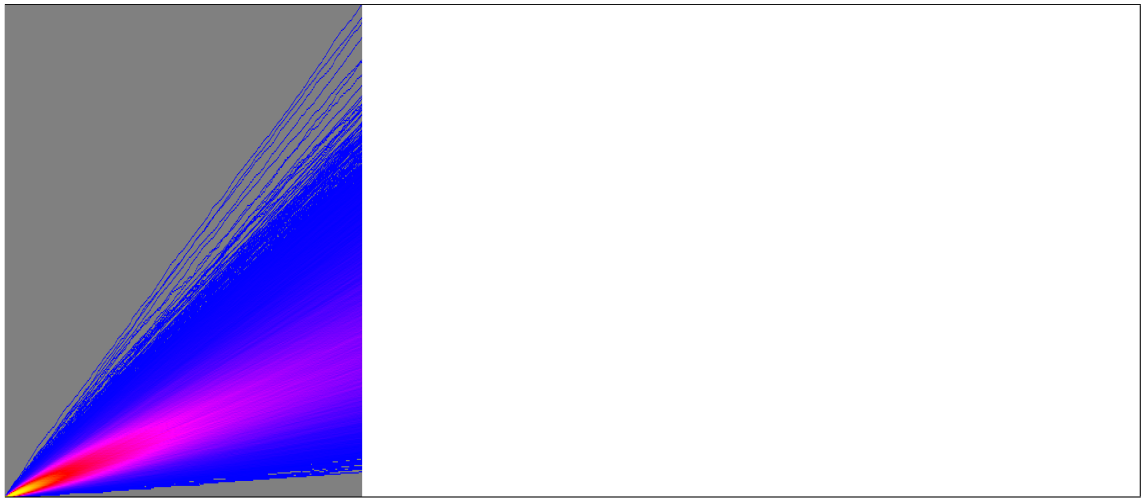


Figure A.62: From Fig. 4.7 with $P = 0$, $C = 6$, $B = 0$

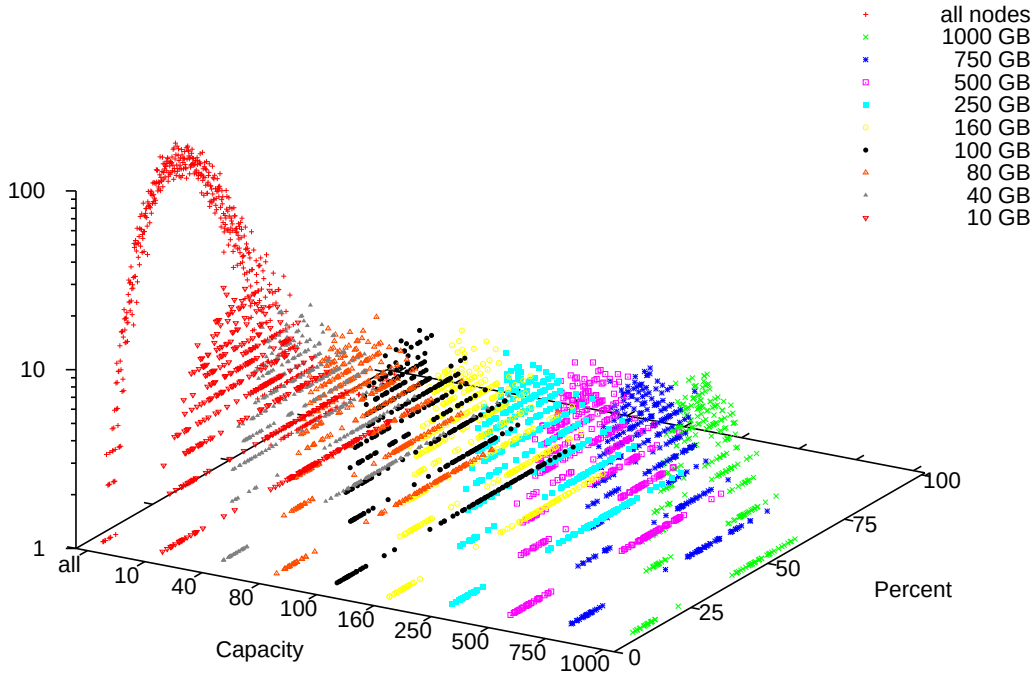


Figure A.63: Node level distribution of Fig. A.62

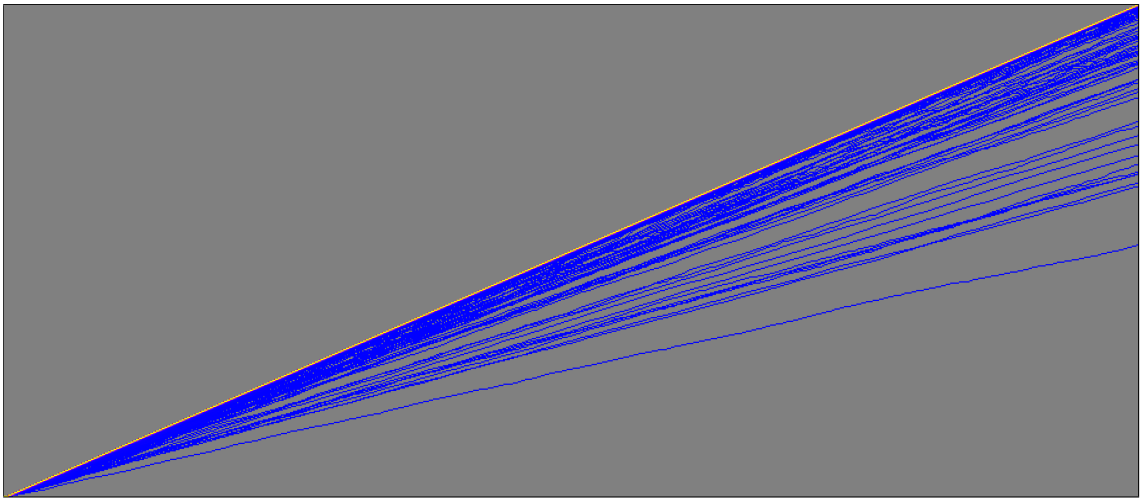


Figure A.64: From Fig. 4.7 with $P = 0$, $C = 6$, $B = 1$

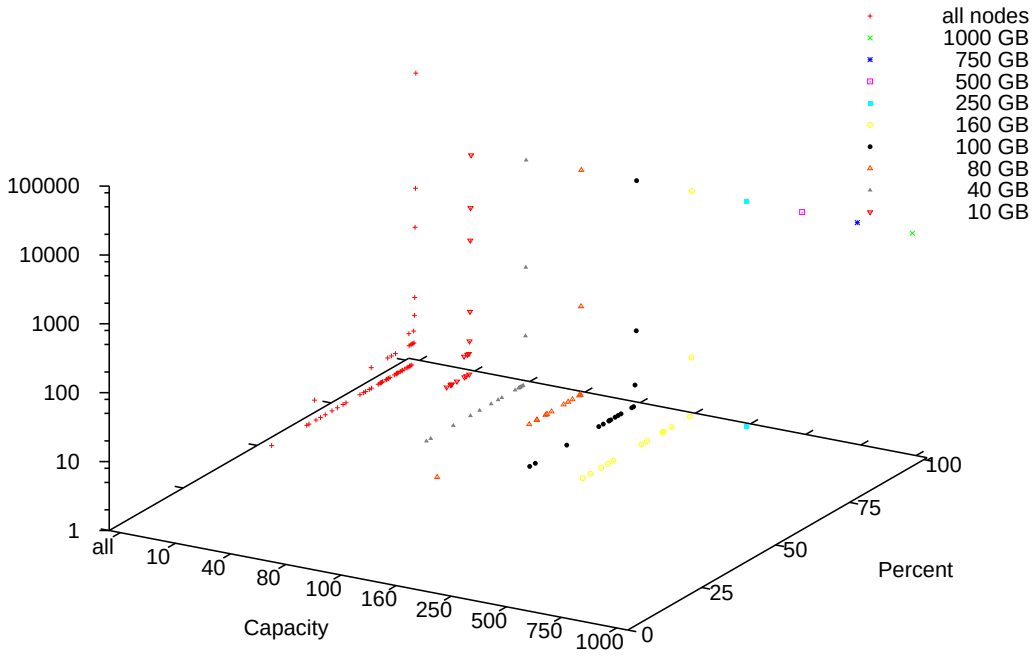


Figure A.65: Node level distribution of Fig. A.64

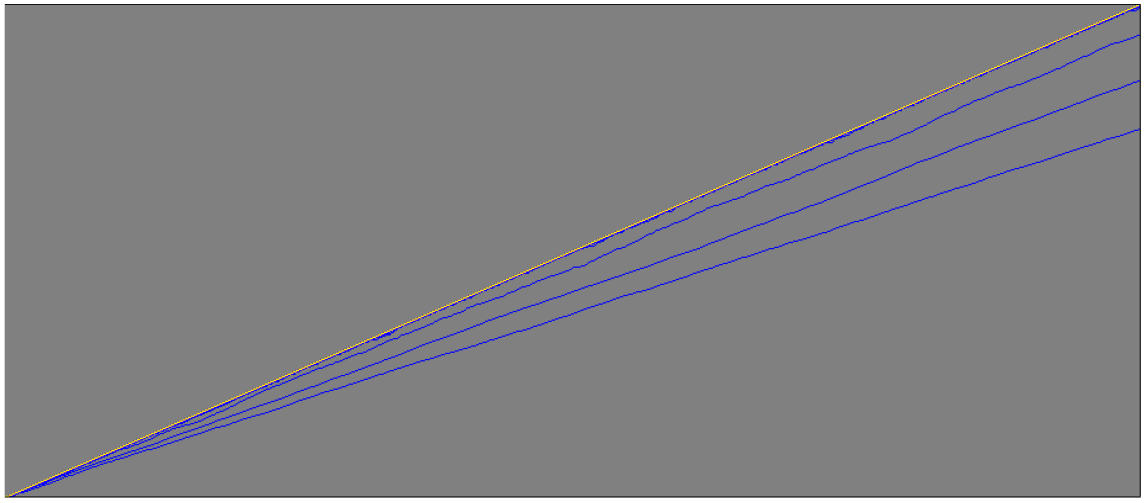


Figure A.66: From Fig. 4.7 with $P = 0$, $C = 6$, $B = 2$

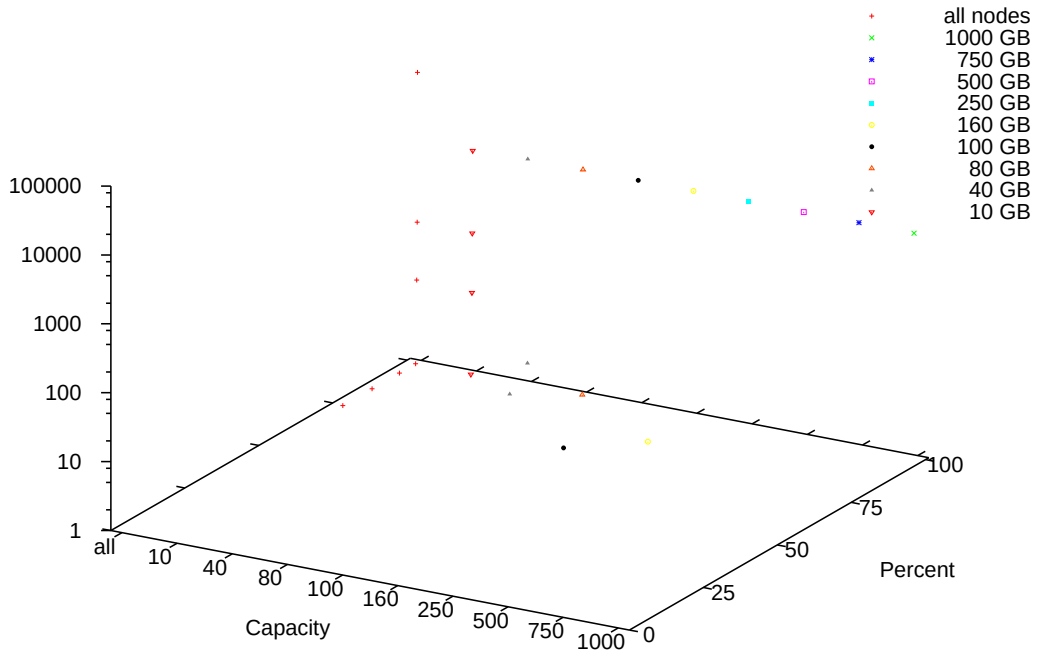


Figure A.67: Node level distribution of Fig. A.66

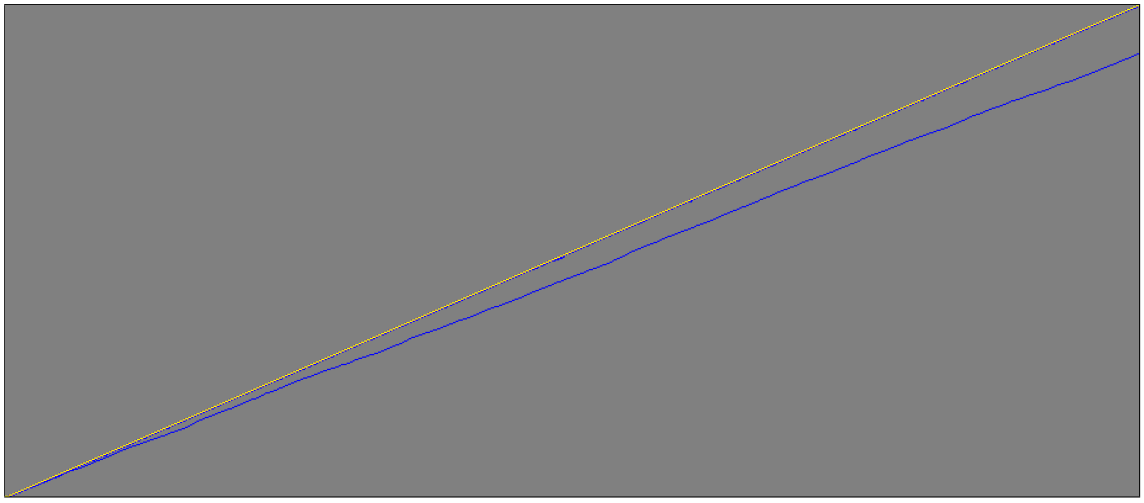


Figure A.68: From Fig. 4.7 with $P = 0$, $C = 6$, $B = 3$

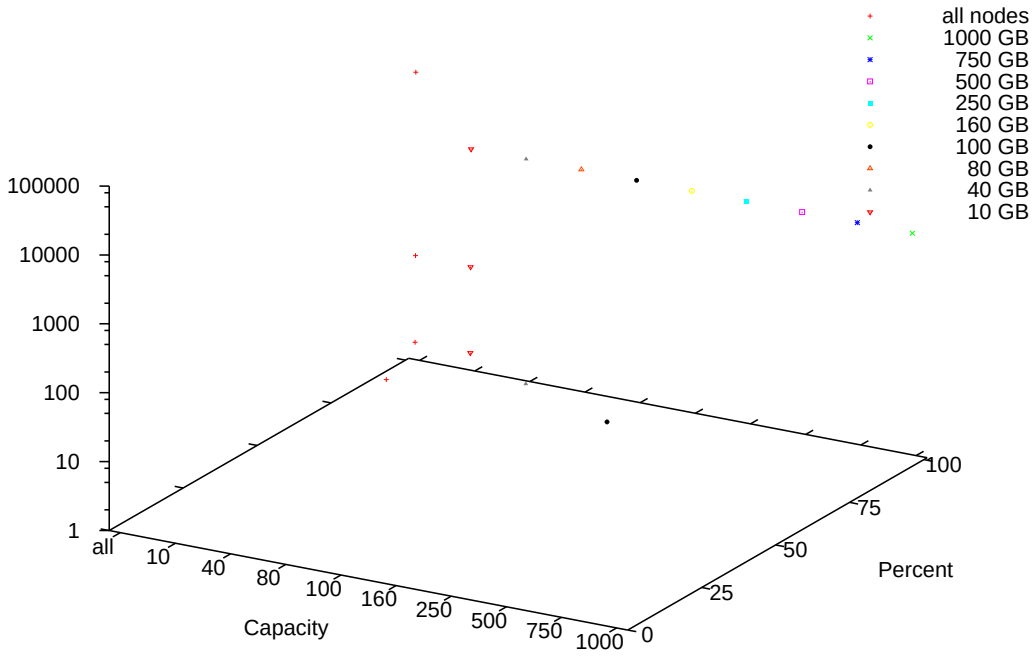


Figure A.69: Node level distribution of Fig. A.68

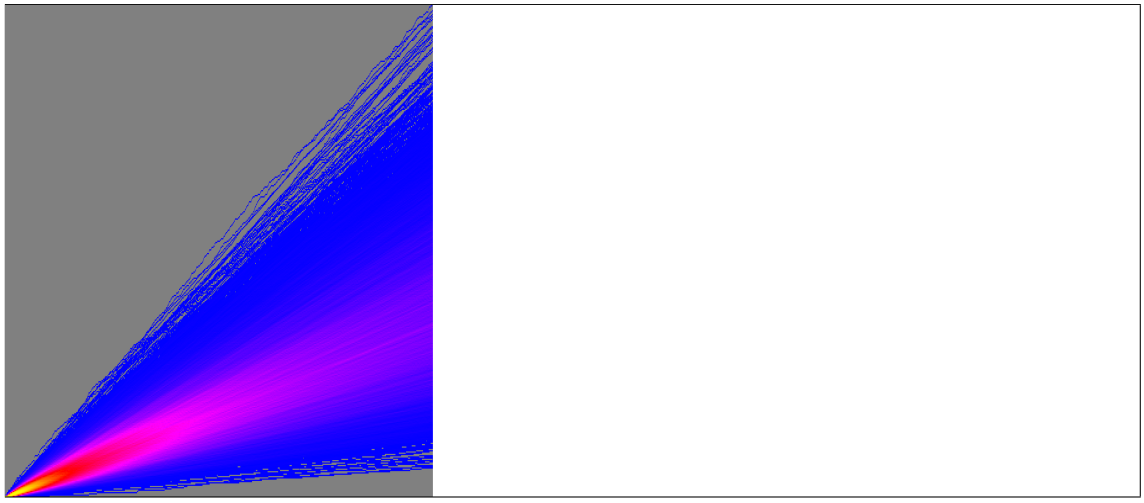


Figure A.70: From Fig. 4.7 with $P = 0$, $C = 7$, $B = 0$

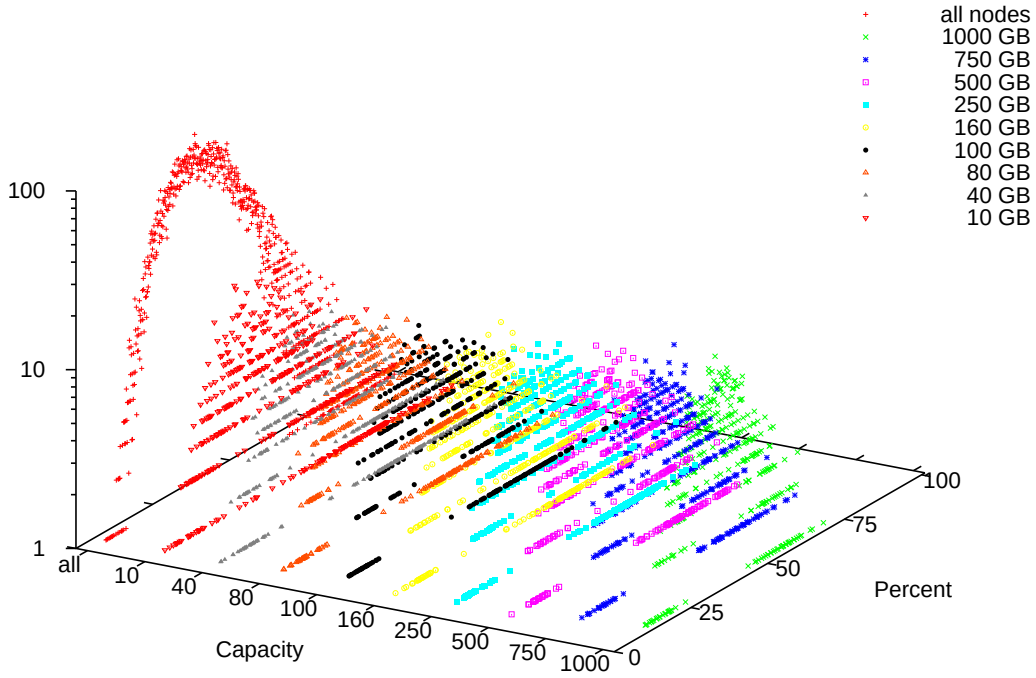


Figure A.71: Node level distribution of Fig. A.70

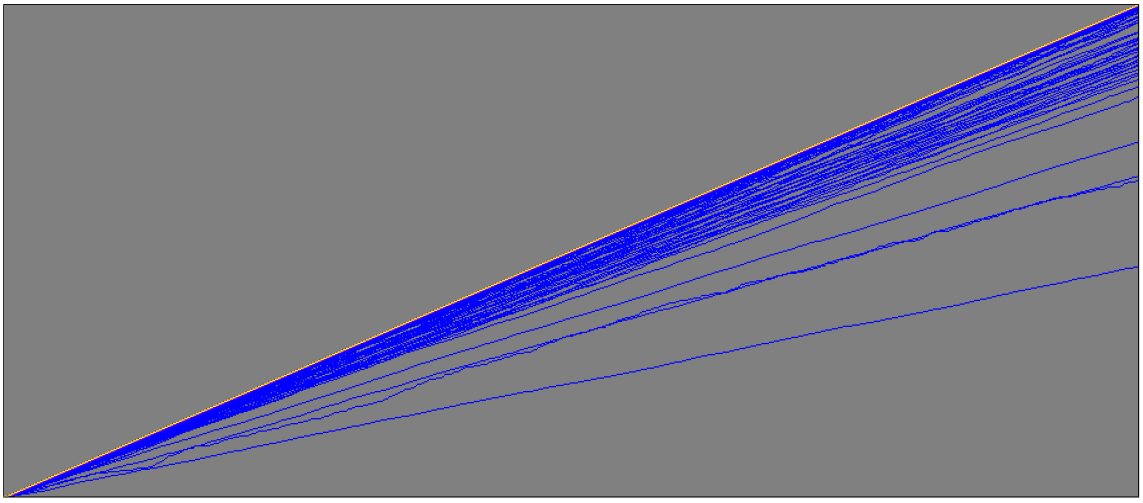


Figure A.72: From Fig. 4.7 with $P = 0$, $C = 7$, $B = 1$

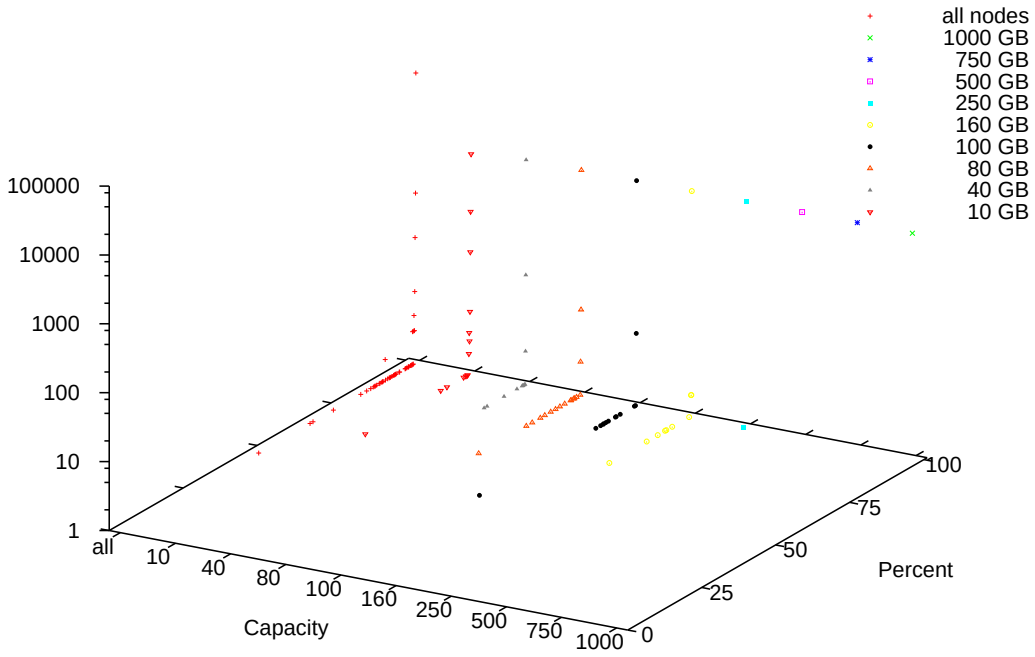


Figure A.73: Node level distribution of Fig. A.72

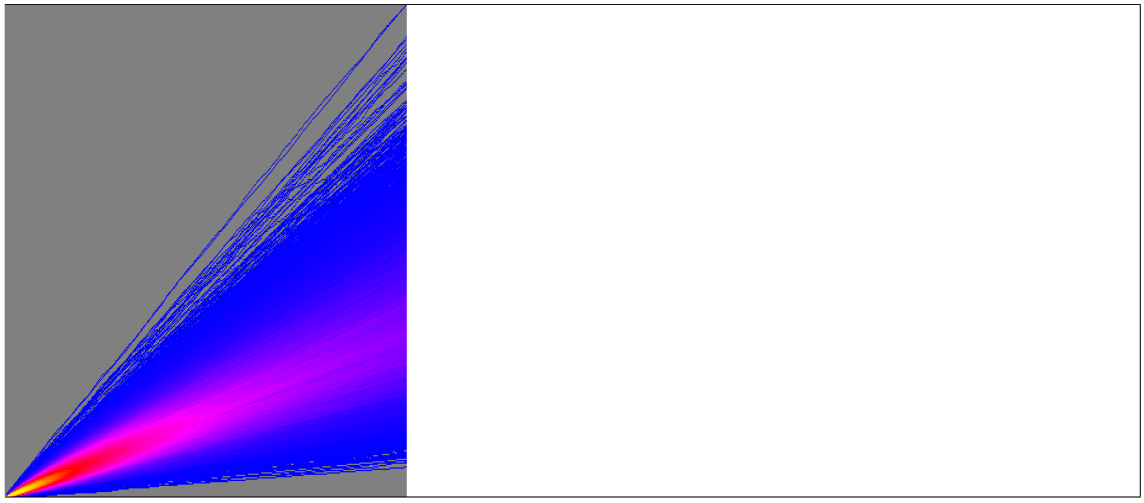


Figure A.74: From Fig. 4.7 with $P = 0$, $C = 8$, $B = 0$

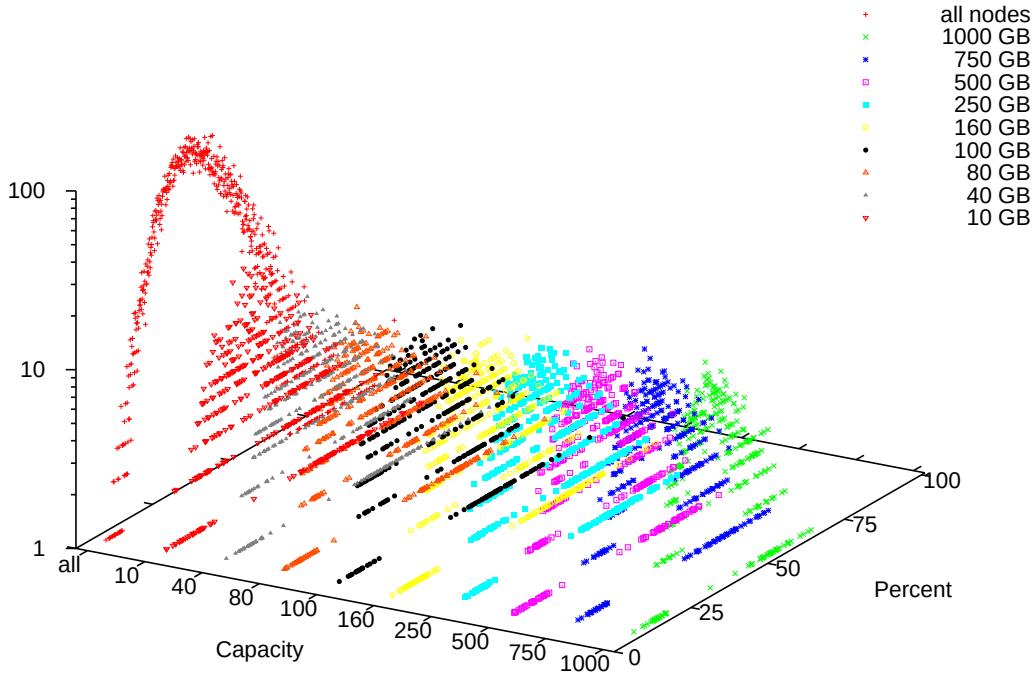


Figure A.75: Node level distribution of Fig. A.74

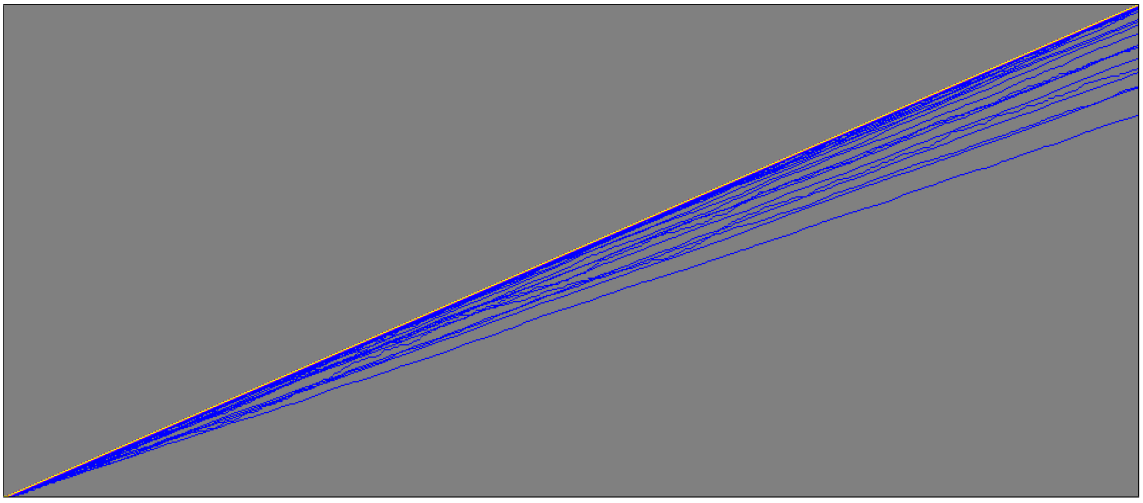


Figure A.76: From Fig. 4.7 with $P = 0$, $C = 8$, $B = 1$

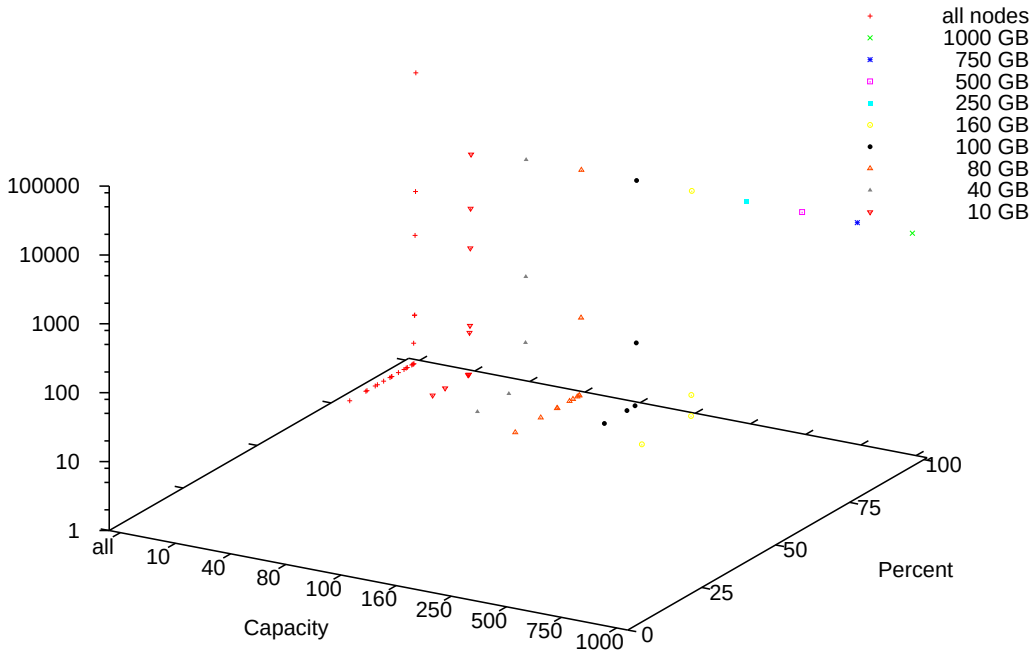


Figure A.77: Node level distribution of Fig. A.76

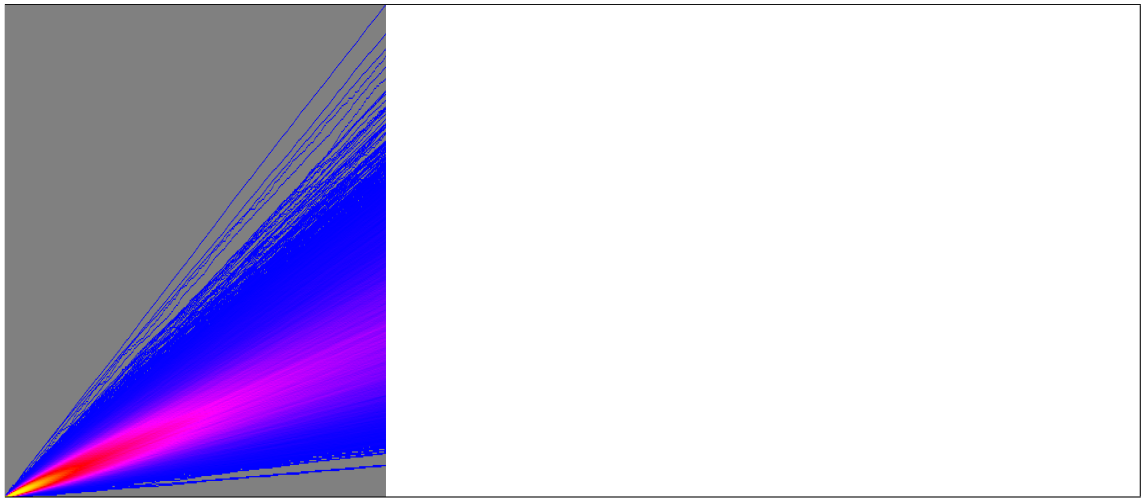


Figure A.78: From Fig. 4.8 with $P = 0$, $C = 9$, $B = 0$

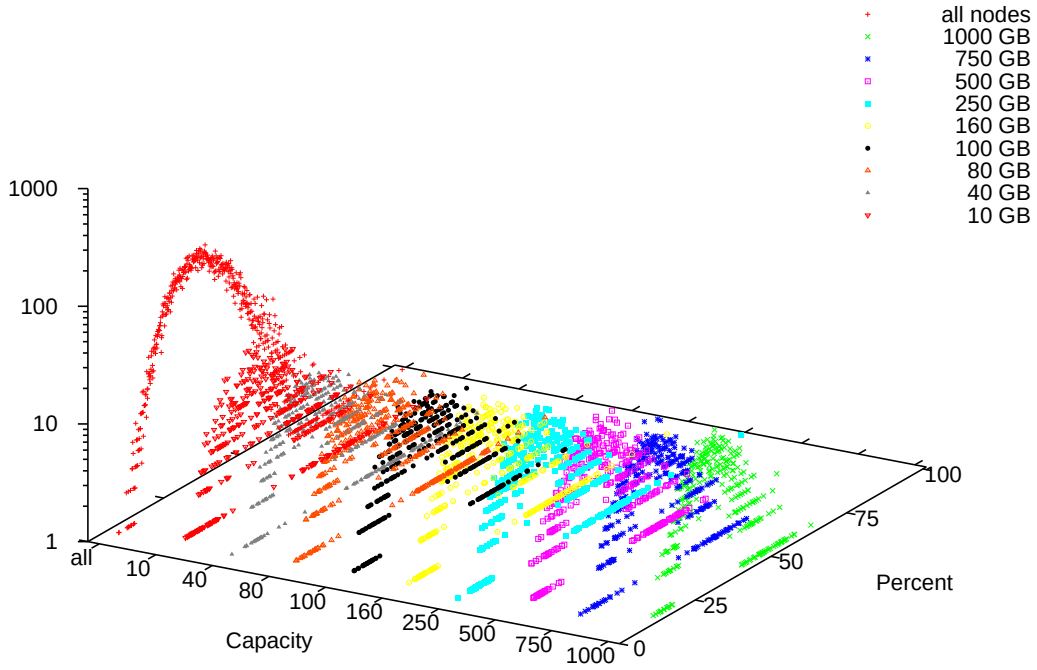


Figure A.79: Node level distribution of Fig. A.78

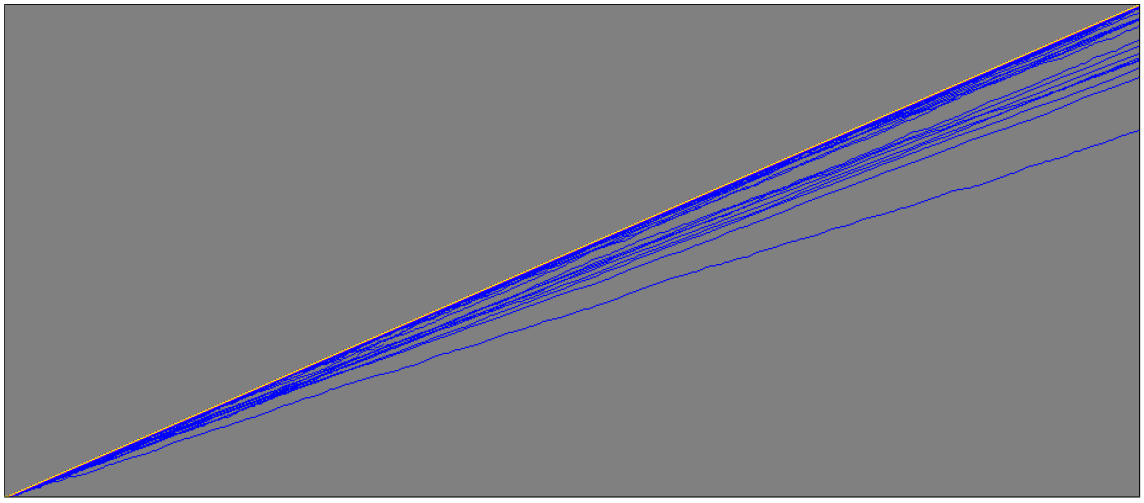


Figure A.80: From Fig. 4.8 with $P = 0$, $C = 9$, $B = 1$

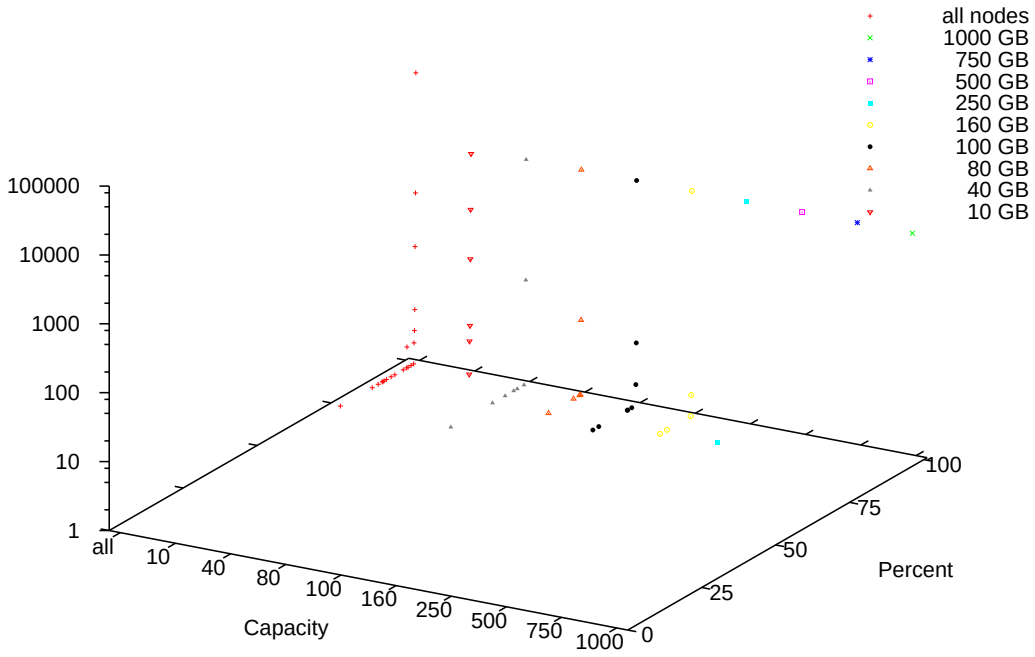


Figure A.81: Node level distribution of Fig. A.80

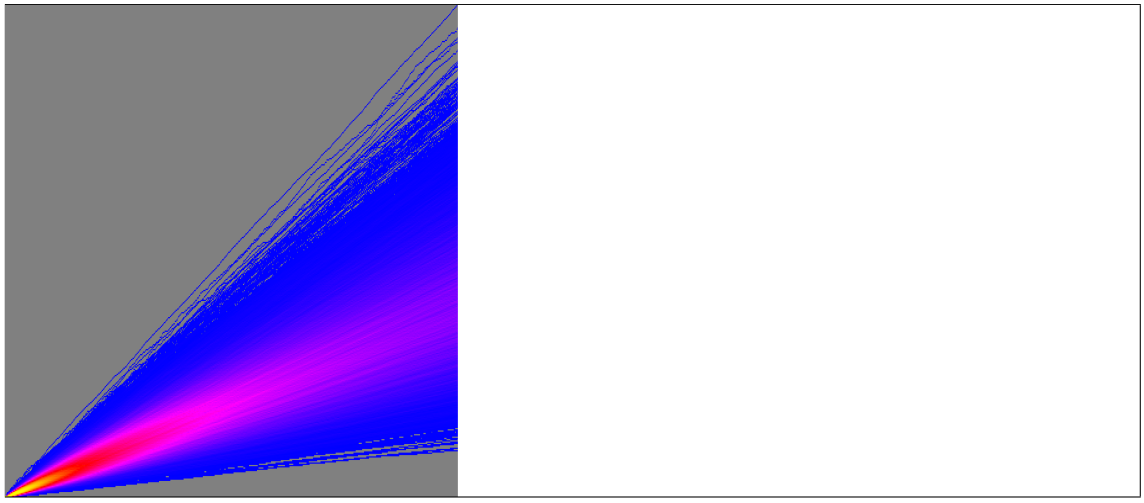


Figure A.82: From Fig. 4.8 with $P = 0$, $C = 10$, $B = 0$

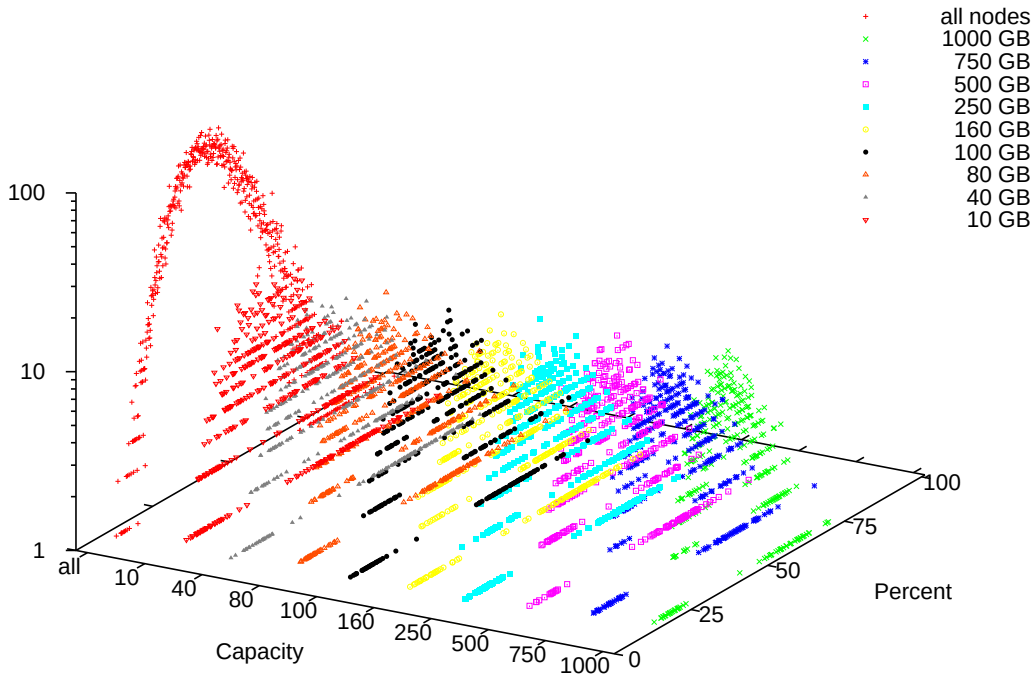


Figure A.83: Node level distribution of Fig. A.82

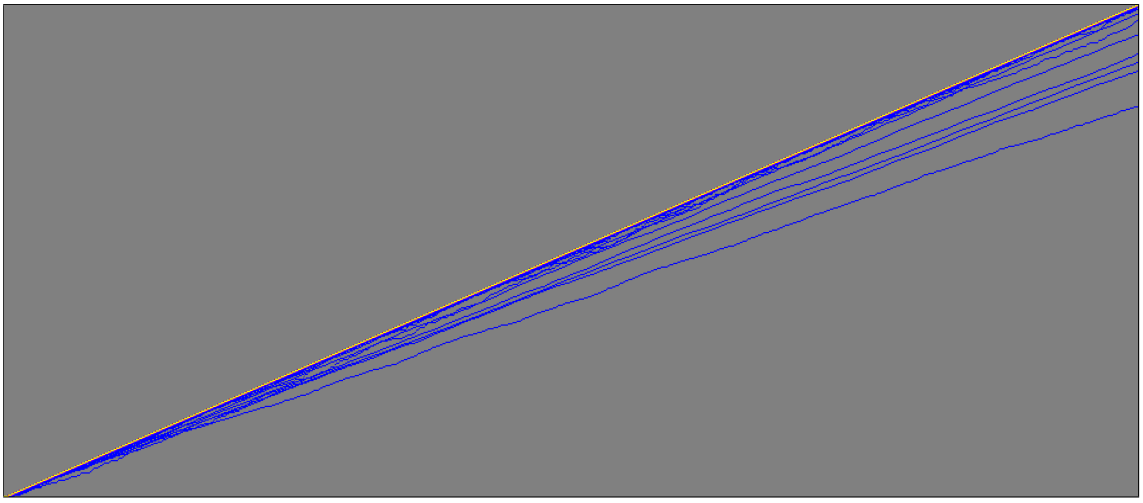


Figure A.84: From Fig. 4.8 with $P = 0$, $C = 10$, $B = 1$

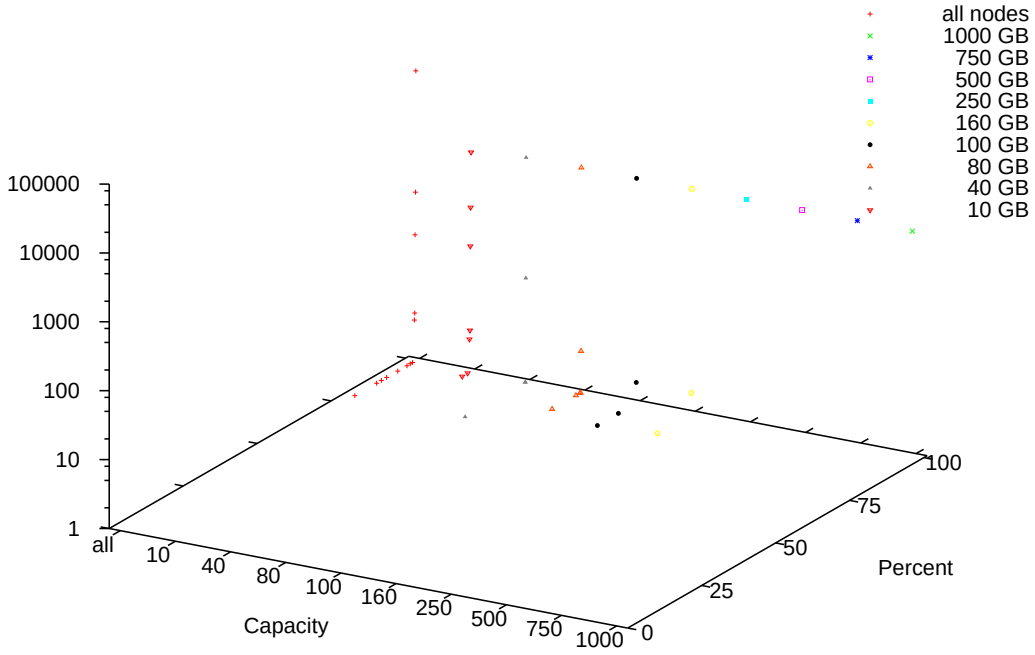


Figure A.85: Node level distribution of Fig. A.84

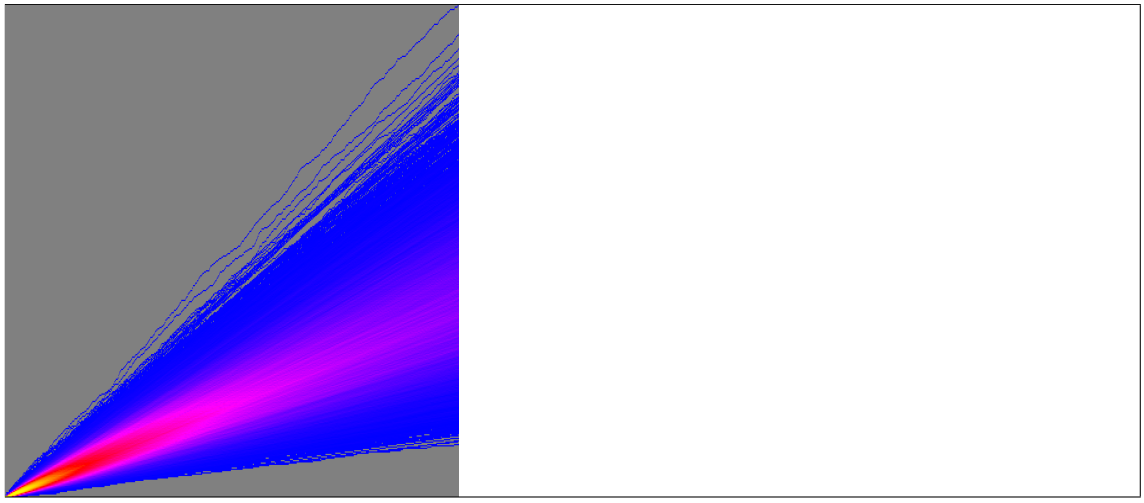


Figure A.86: From Fig. 4.8 with $P = 0$, $C = 11$, $B = 0$

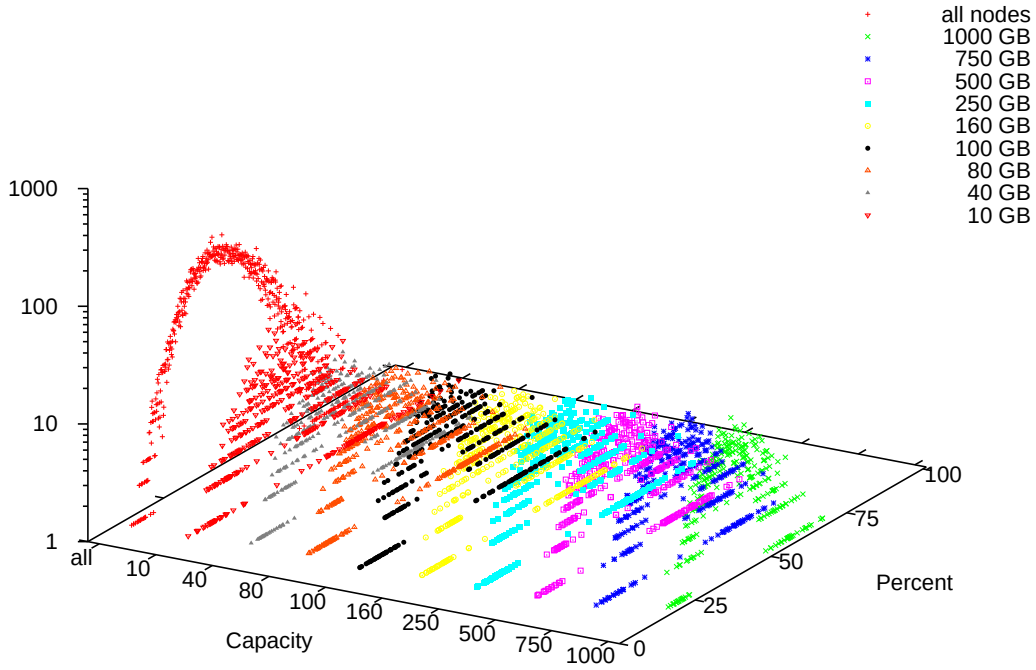


Figure A.87: Node level distribution of Fig. A.86

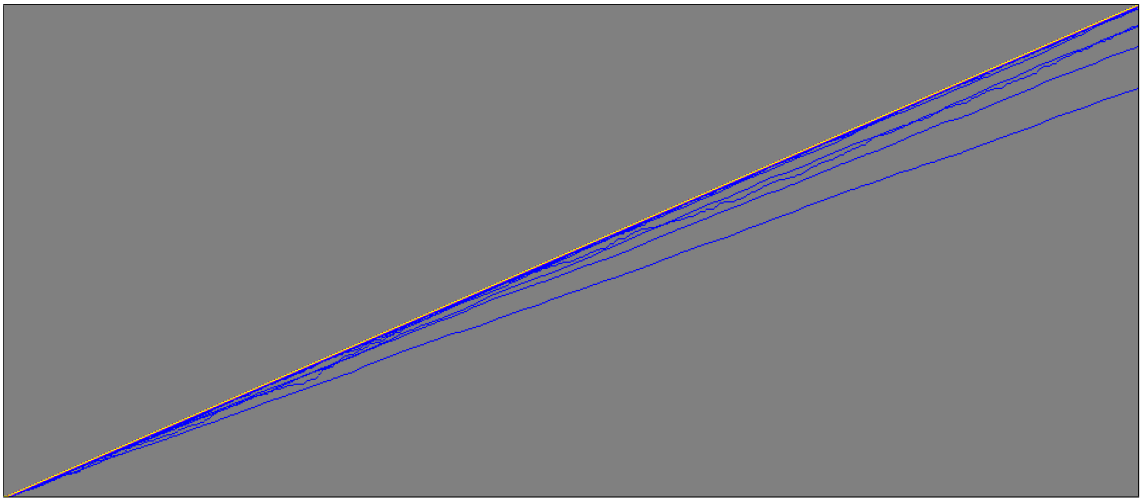


Figure A.88: From Fig. 4.8 with $P = 0$, $C = 11$, $B = 1$

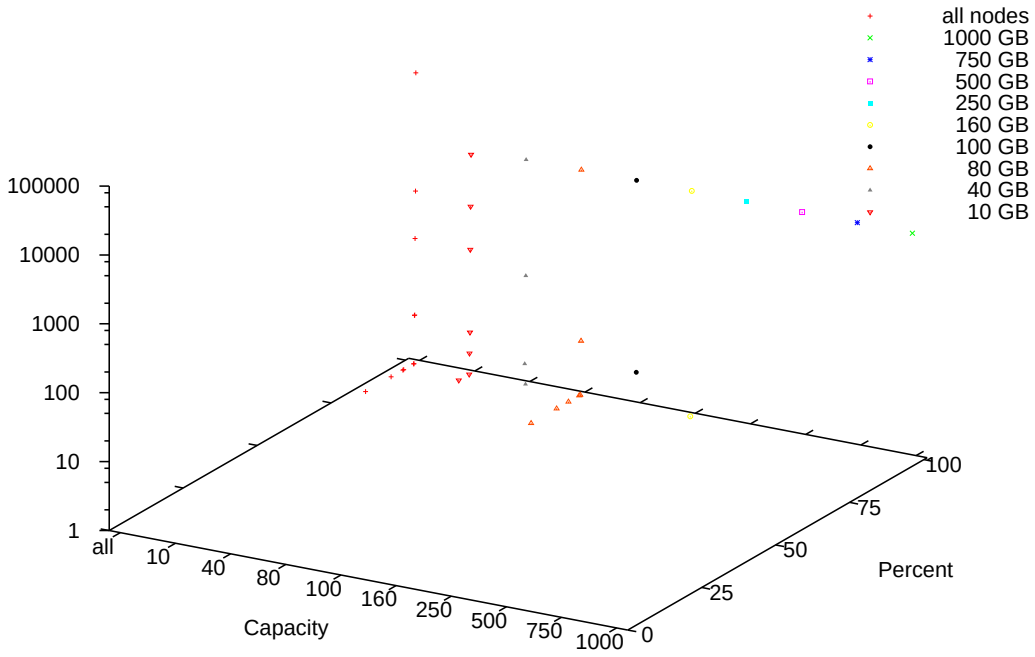


Figure A.89: Node level distribution of Fig. A.88

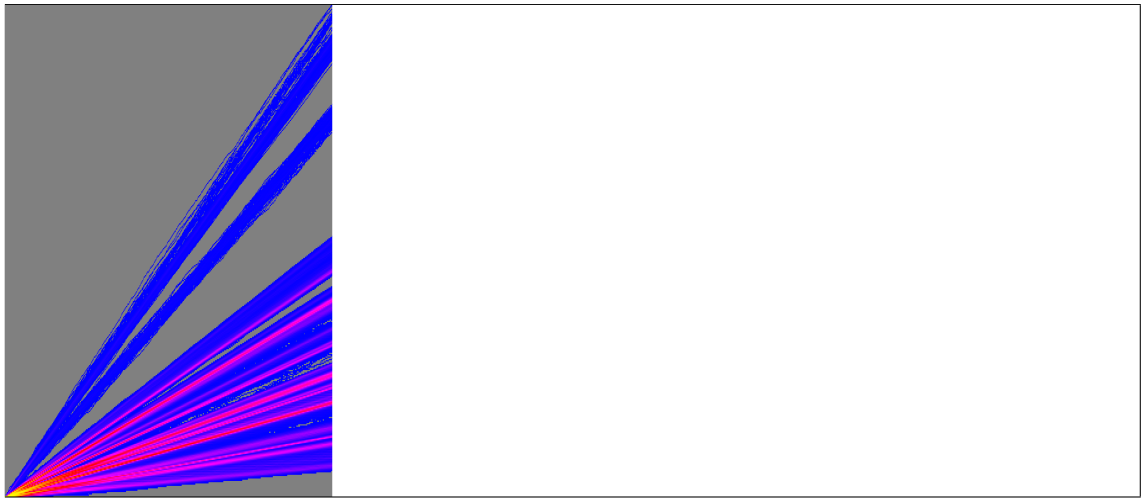


Figure A.90: From Fig. 4.9 with $P = 0$, $C = 0$, $B = 0$

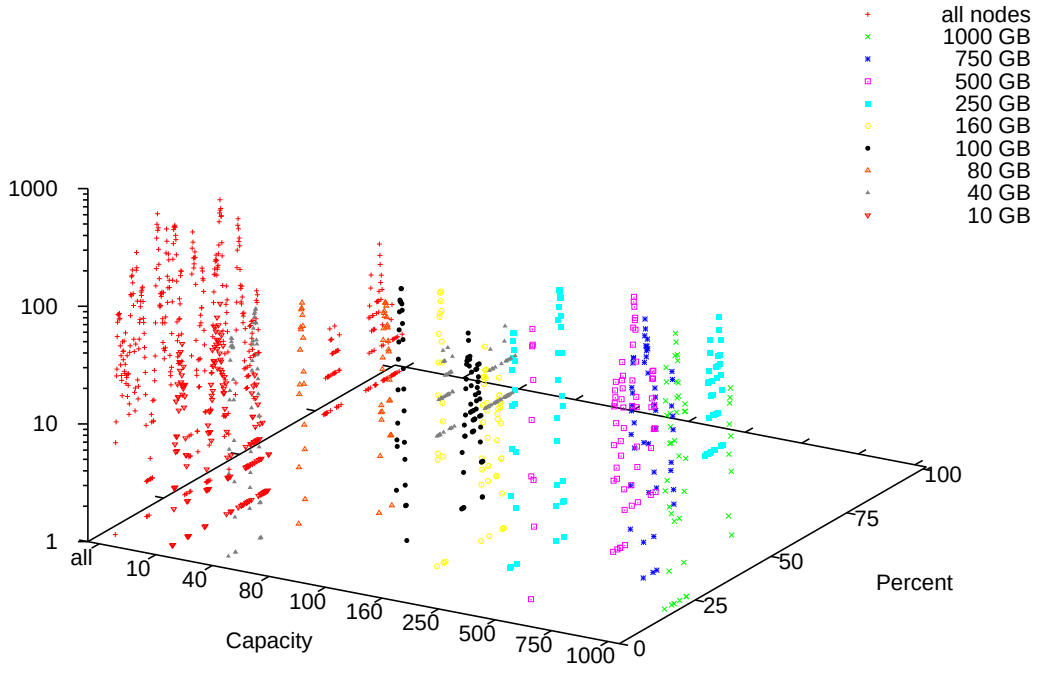


Figure A.91: Node level distribution of Fig. A.90

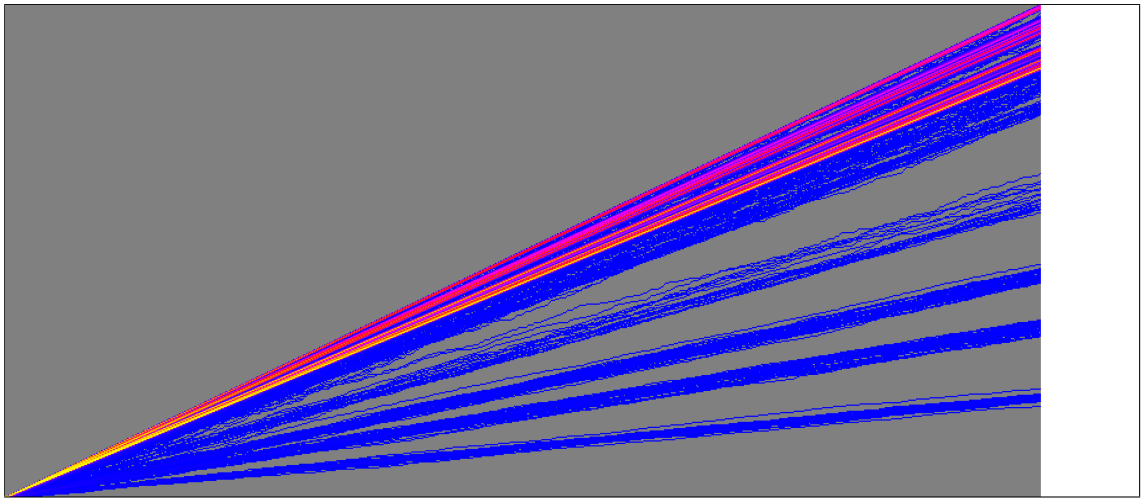


Figure A.92: From Fig. 4.9 with $P = 0$, $C = 0$, $B = 1$

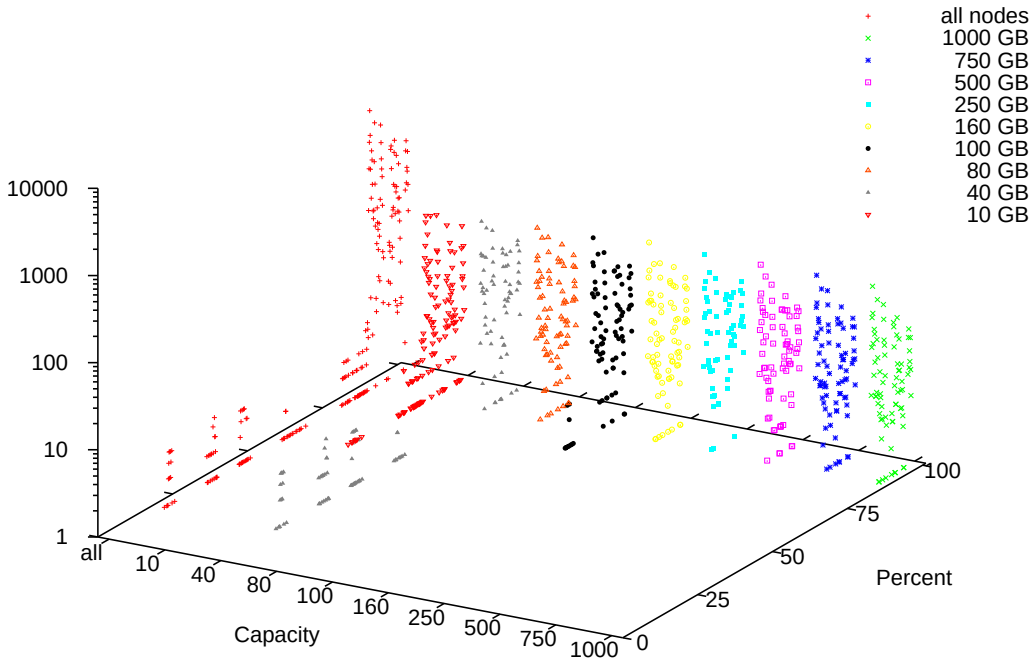


Figure A.93: Node level distribution of Fig. A.92

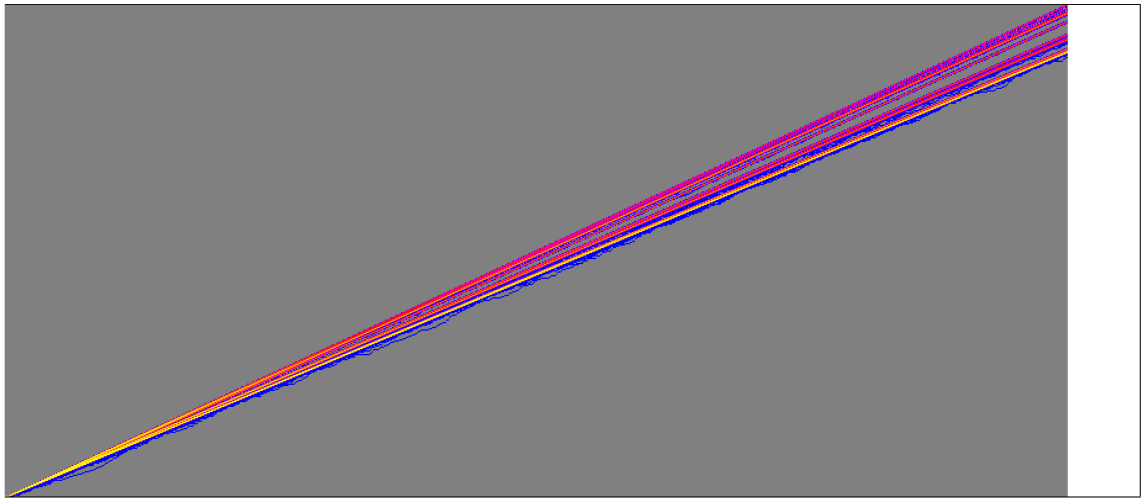


Figure A.94: From Fig. 4.9 with $P = 0$, $C = 0$, $B = 2$

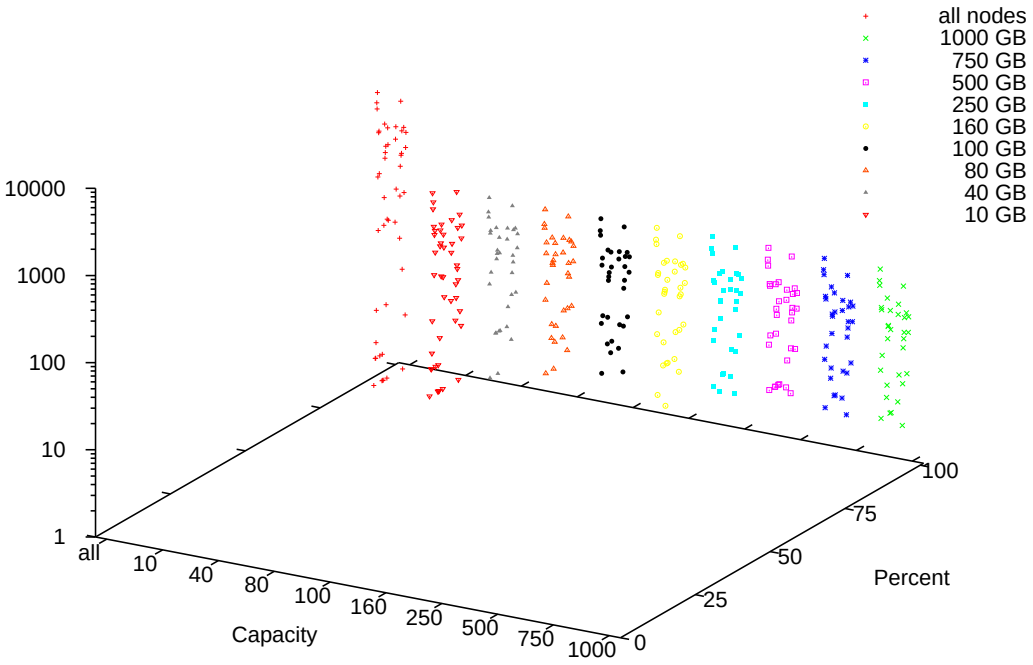


Figure A.95: Node level distribution of Fig. A.94

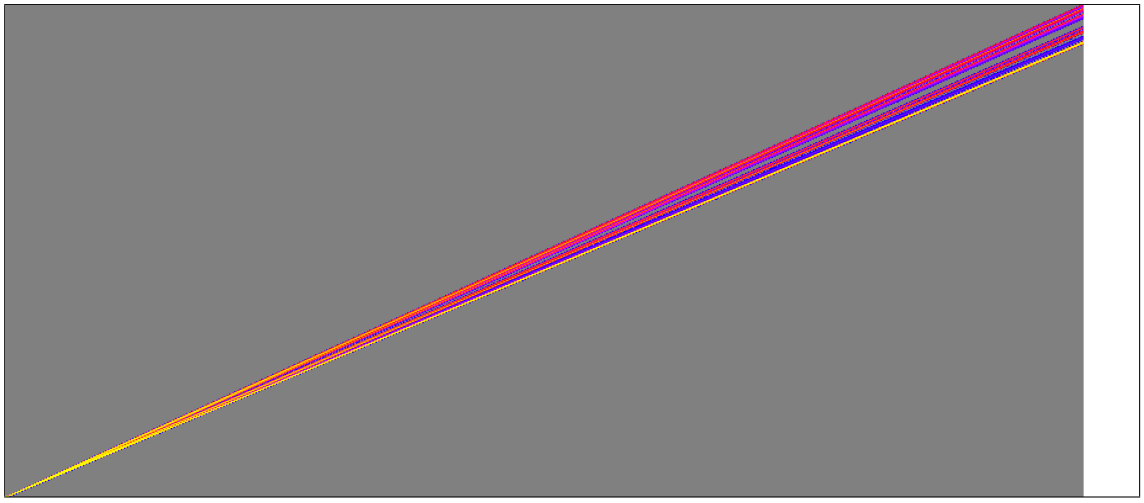


Figure A.96: From Fig. 4.9 with $P = 0$, $C = 0$, $B = 3$

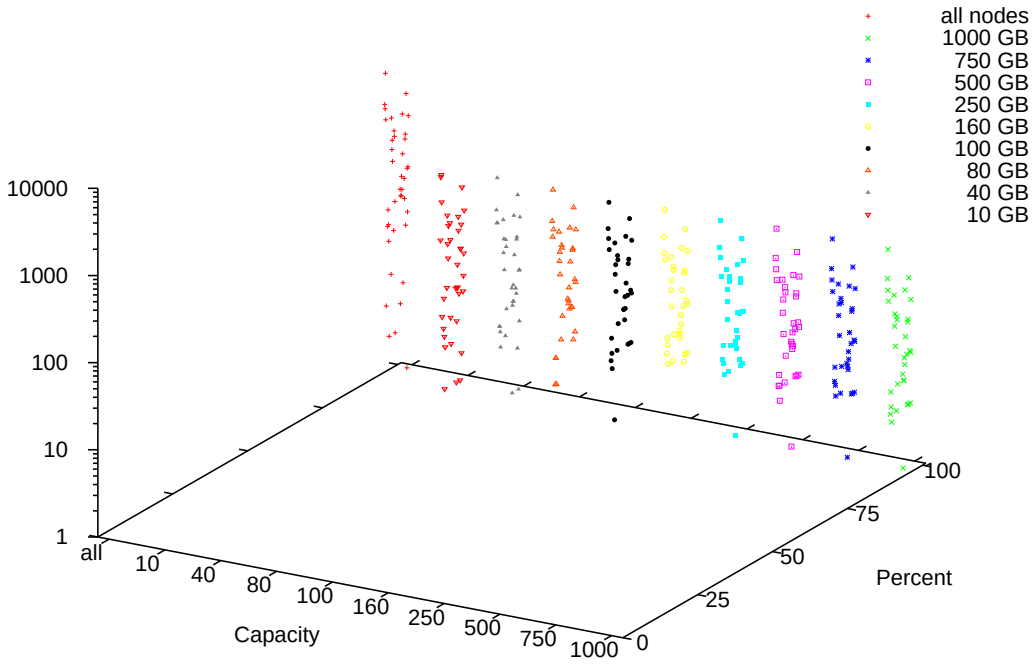


Figure A.97: Node level distribution of Fig. A.96

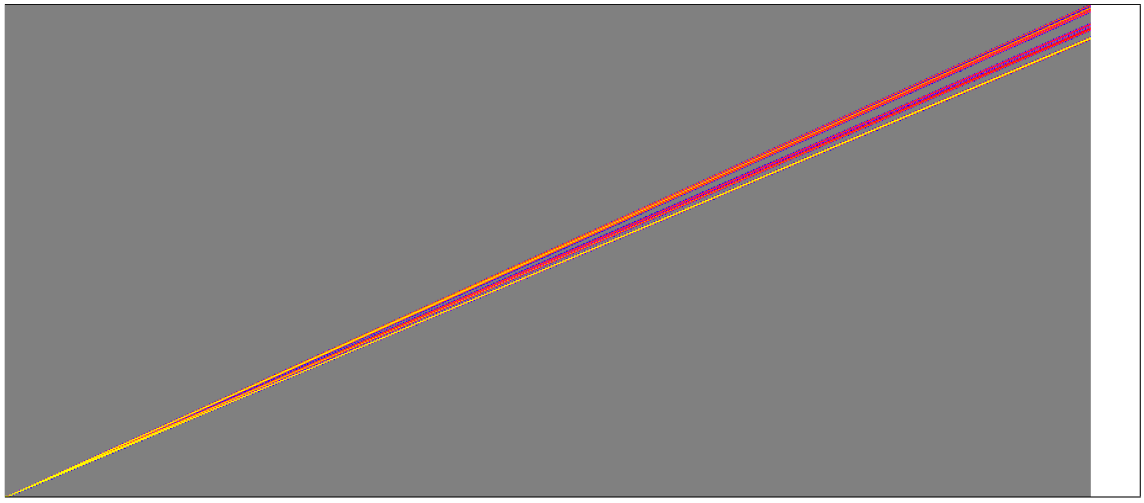


Figure A.98: From Fig. 4.9 with $P = 0$, $C = 0$, $B = 4$

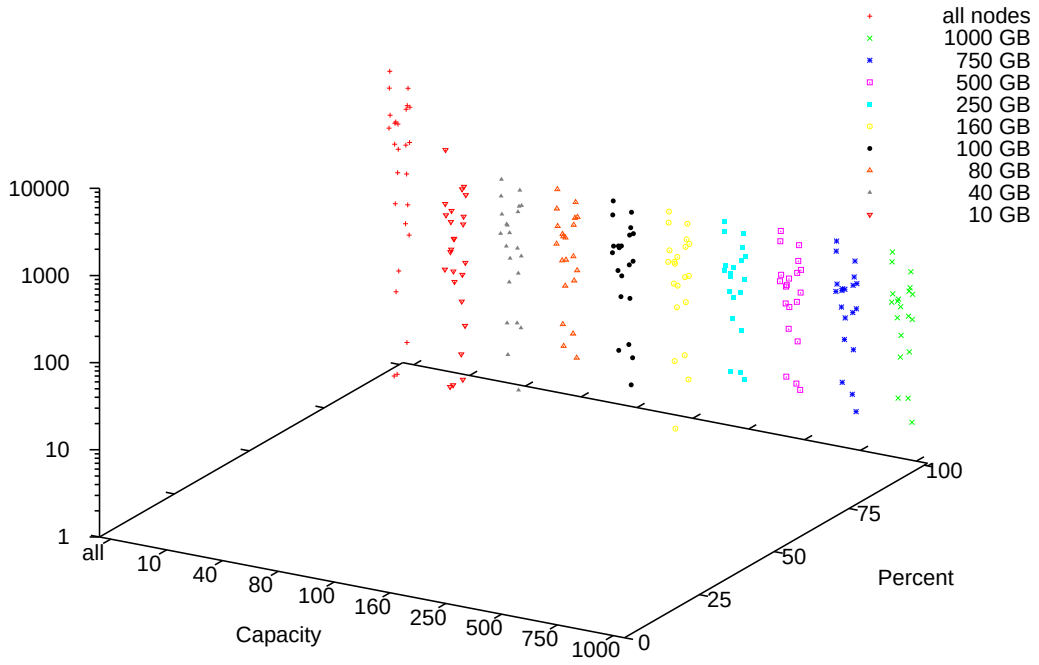


Figure A.99: Node level distribution of Fig. **??**

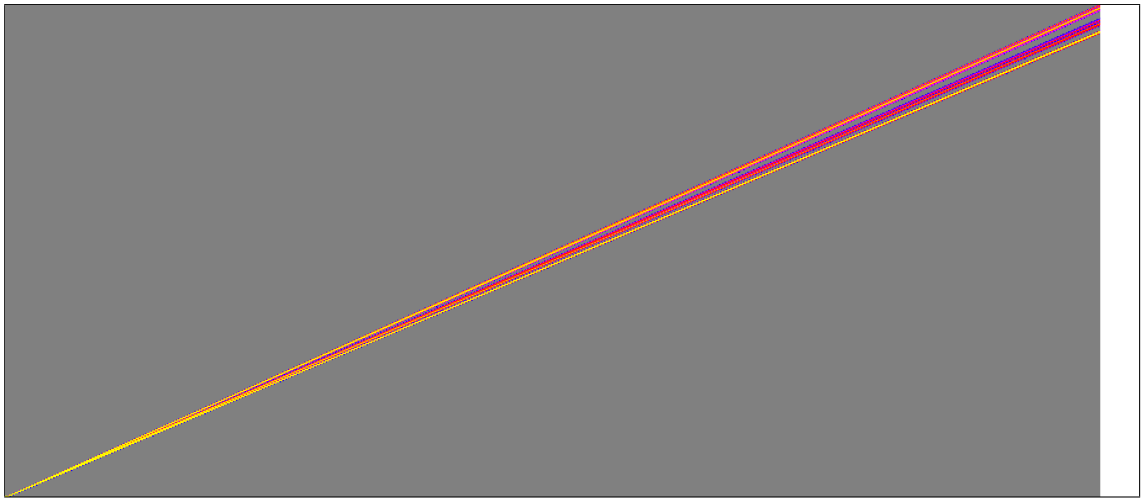


Figure A.100: From Fig. 4.9 with $P = 0, C = 0, B = 5$

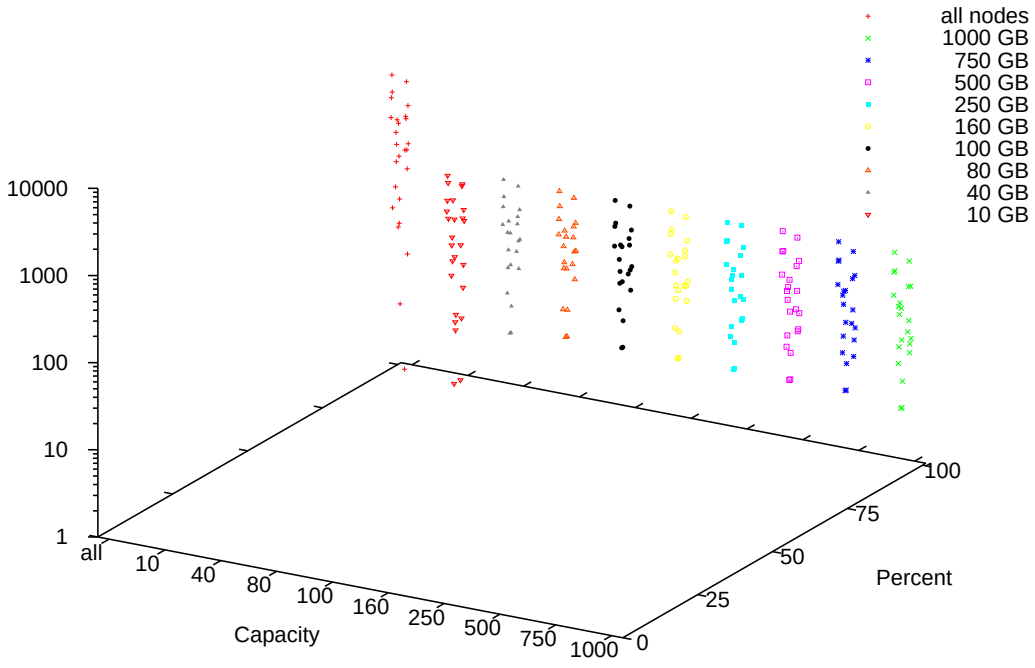


Figure A.101: Node level distribution of Fig. A.100

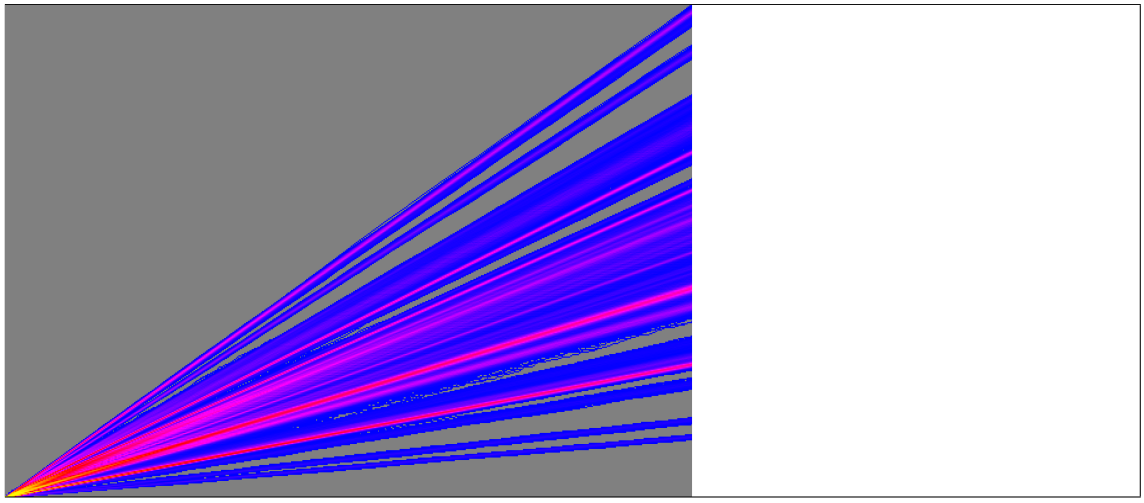


Figure A.102: From Fig. 4.9 with $P = 0$, $C = 2$, $B = 0$

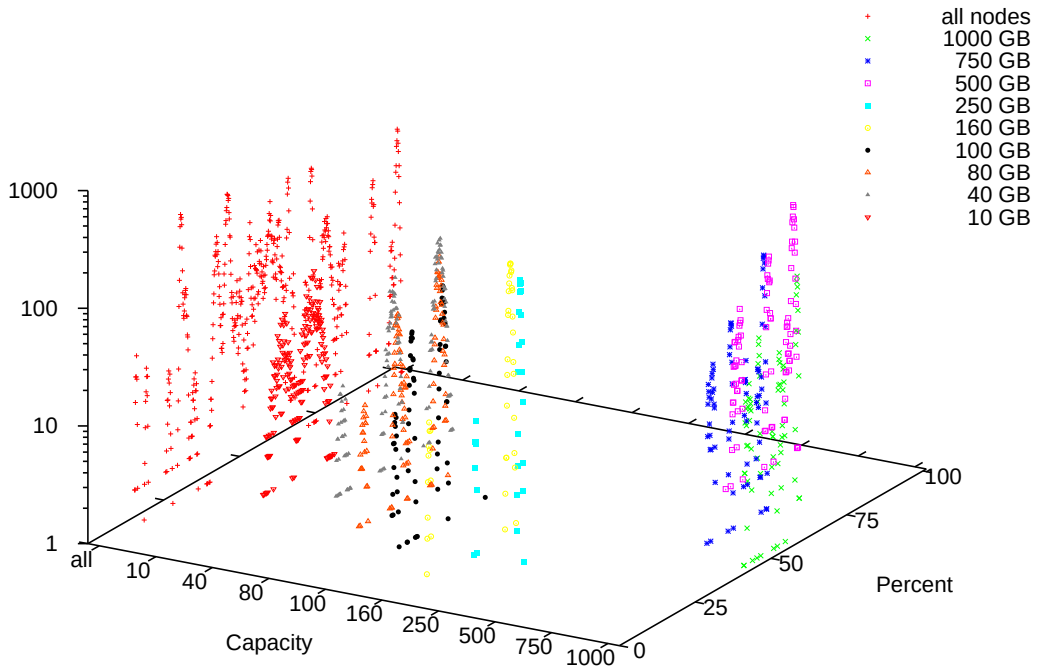


Figure A.103: Node level distribution of Fig. A.102

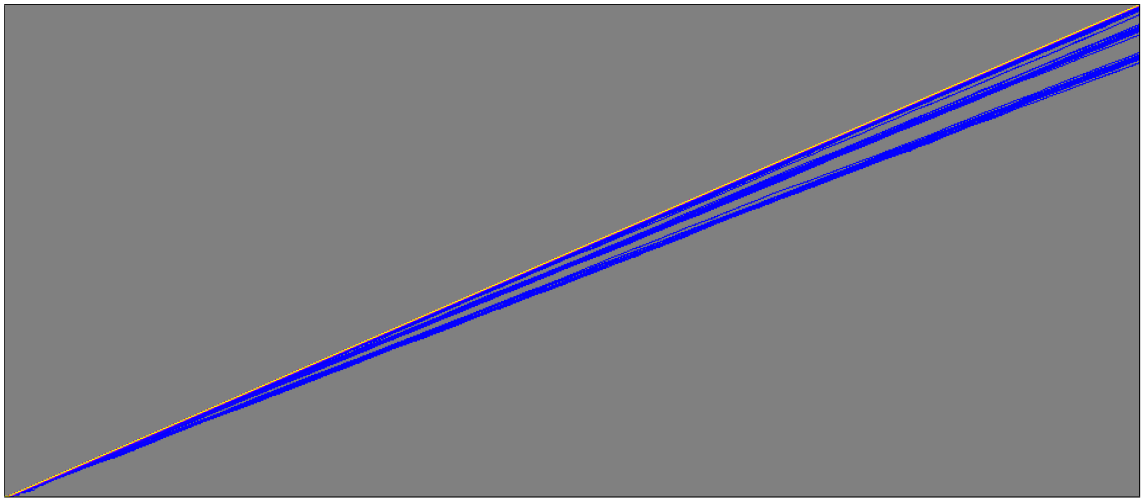


Figure A.104: From Fig. 4.9 with $P = 0$, $C = 2$, $B = 1$

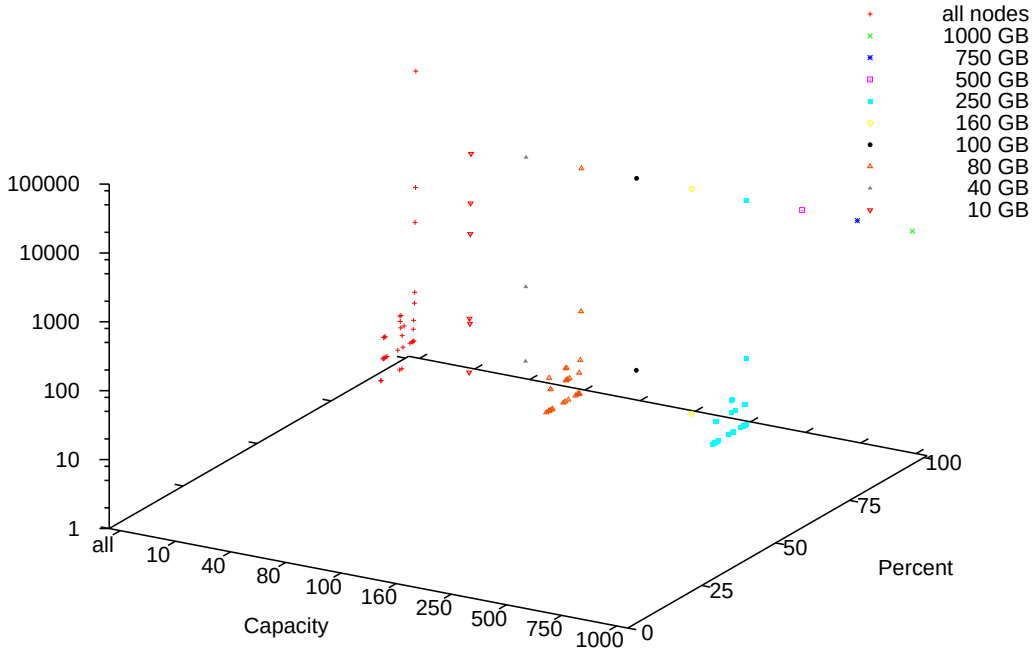


Figure A.105: Node level distribution of Fig. A.104

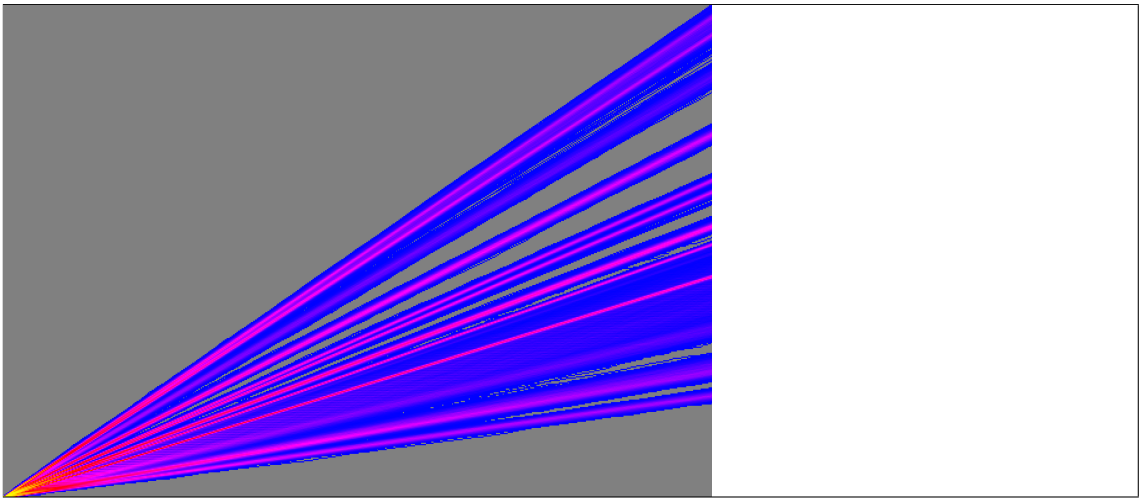


Figure A.106: From Fig. 4.9 with $P = 0$, $C = 5$, $B = 0$

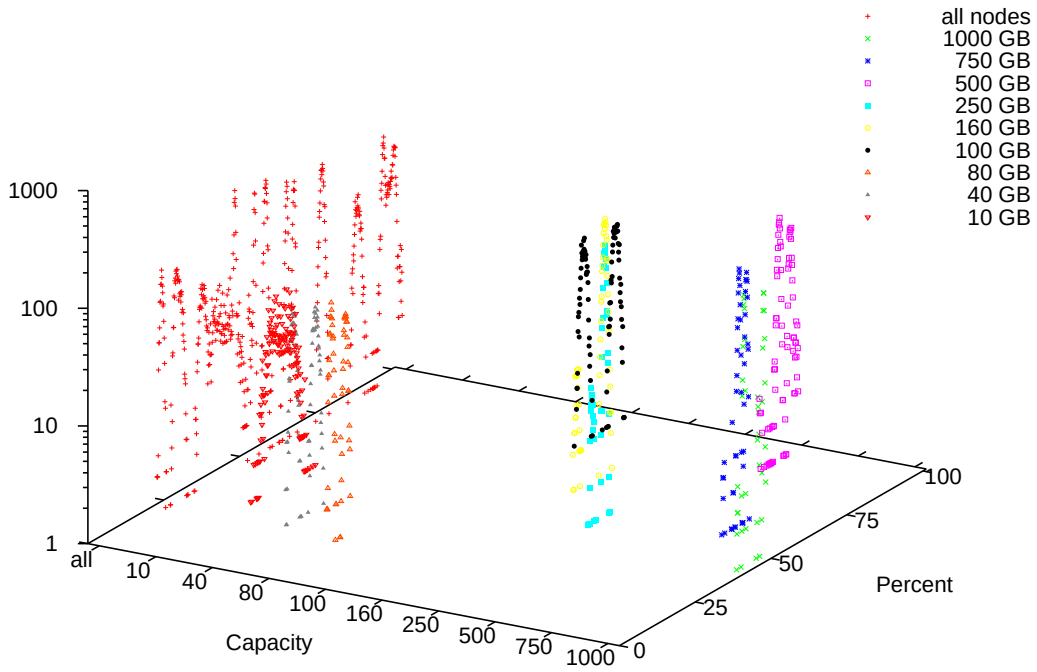


Figure A.107: Node level distribution of Fig. A.106

List of Figures

Bibliography

[1] P. K. Agarwal and M. Sharir. Davenport–schinzel sequences and their geometric applications. Technical Report Technical report DUKE–TR–1995–21, 1995.

[2] P. K. Agarwal and M. Sharir. Simple bounds. In *Davenport-Schinzel Sequences and Their Geometric Applications*, chapter 1, pages 1–47. Cambridge University Press, in handbook of computational geometry edition, 1995. ISBN 0-521-47025-0.

[3] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking. Balanced Allocations: The Heavily Loaded Case. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 745–754, New York, NY, USA, 2000. ACM.

[4] A. Bhargava, K. Kothapalli, C. Riley, C. Scheideler, and M. Thober. Pagoda: a dynamic overlay network for routing, data management, and multicasting. In *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 170–179, New York, NY, USA, 2004. ACM Press.

[5] P. Bleckmann, S. Bötcher, E. Cesnavicius, T. Hollerung, B. Kühnel, M. Jing Liu, S. Obermeier, S. Oberthür, F. Peter, F. J. Rammig, C. Schindelhauer, G. Schomaker, T. Steenweg, Q. A. Tarar, M. Tiemeyer, A. Thürling, and A. Vater. The design of pamanet the paderborn mobile ad-hoc network, extended abstract. Technical report, Heinz Nixdorf Institut, 2004.

[6] P. Bleckmann, G. Schomaker, and A. Slowik. Virtualization with prefetching abilities based on iscsi. In *Proceeding of International Workshop on Storage Network Architecture and Parallel I/O*, number 2, pages 40–47. ACM Press, New York, NY, USA, 2004.

[7] O. Bonorden, J. Gehweiler, and F. Meyer auf der Heide. A web computing environment for parallel algorithms in Java. In *Proceeedings of Interna-*

tional Conference on Parallel Processing and Applied Mathematics (PPAM), Poznan, Poland, September 2005.

[8] A. Brinkmann, S. Effert, F. Meyer auf der Heide, and C. Scheideler. Dynamic and redundant data placement. In *27th IEEE International Conference on Distributed Computing Systems (ICDCS 2007)*, Toronto, Canada, 25 - 29 June 2007.

[9] A. Brinkmann, M. Heidebuer, F. Meyer auf der Heide, U. Rückert, K. Salzwedel, and M. Vodisek. V:drive – costs and benefits of an out–of–band storage virtualization system. In *Proceedings of the 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies (MSST)*, pages 153 – 157, College Park, Maryland, USA, 13 - 16 Apr. 2004.

[10] A. Brinkmann, F. Meyer auf der Heide, K. Salzwedel, C. Scheideler, M. Vodisek, and U. Rückert. Storage management as means to cope with exponential information growth. In *Proceedings of SSGRR 2003*, L'Aquila, Italy, 28 July - 3 Aug. 2003.

[11] A. Brinkmann, K. Salzwedel, and C. Scheideler. Efficient, distributed data placement strategies for storage area networks (extended abstract). In *Proceedings of the twelfth annual ACM symposium on Parallel algorithms and architectures*, pages 119–128. ACM Press, 2000.

[12] A. Brinkmann, K. Salzwedel, and C. Scheideler. Compact, adaptive placement schemes for non-uniform distribution requirements. In *Proc. of the 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 53–62, Winnipeg, Manitoba, Canada, 11 - 13 Aug. 2002.

[13] T. Cortes and J. Labarta. A case for heterogeneous disk arrays. In *Proceedings. IEEE International Conference on Cluster Computing.*, pages 319–325, September 2000.

[14] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 202–215, New York, NY, USA, 2001. ACM.

[15] P. Druschel and A. Rowstron. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In R. Guerraoui, editor, *Middleware 2001, IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001, Proceedings*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer, 2001.

[16] J. Gehweiler and G. Schomaker. Distributed load balancing in heterogeneous peer-to-peer networks for web computing libraries. In *DS-RT '06: Proceedings of the 10th IEEE international symposium on Distributed Simulation and Real-Time Applications*, pages 51–62, Washington, DC, USA, 2006. IEEE Computer Society.

[17] P. Godfrey and I. Stoica. Heterogeneity and Load Balance in Distributed Hash Tables. *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, 1:596–606 vol. 1, 13-17 March 2005.

[18] P. B. Godfrey. Balls and bins with structure: balanced allocations on hypergraphs. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 511–517, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[19] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed object location in a dynamic network. In *Proceedings of the 14th Annual ACM Symposium on Parallel ALgorithms and Architectures (SPAA-02)*, pages 41–52, New York, Aug. 10–13 2002. ACM Press.

[20] J. Judd. *Principles of SAN Design: Updated for 2007*. Infinity Publishing, September 2007.

[21] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 654–663, El Paso, Texas, 4–6 May 1997.

[22] D. E. Knuth. *Art of Computer Programming, Volume 3: Sorting and Searching (2nd Edition)*. Addison-Wesley Professional, April 1998.

[23] C. Loeser. *Proaktives Bandbreitenmanagement in heterogenen Content Distribution Netzwerken.* Shaker Verlag, Aachen, November 2007.

[24] C. Loeser, G. Schomaker, A. Brinkmann, M. Vodisek, and M. Heidebuer. Content distribution in video-on-demand p2p networks with arima. In *Proceedings of the 4th International Conference on Networking*, volume 3421, pages 800–810. Springer Verlag, 17 - 21 Apr. 2005.

[25] C. Loeser, G. Schomaker, and M. Schubert. Predictive replication and placement strategies for movie documents in heterogeneous content delivery networks. In *5th International Conference on Networking (ICN).*, Mauritius, 23 - 26 Apr. 2006. ICN, Springer Verlag LNCS.

[26] P. Mahlmann and C. Schindelhauer. *Peer-to-Peer-Netzwerke: Algorithmen und Methoden*, chapter 7, pages 136–140. Springer-Verlag Berlin, 1 edition, 1 June 2007. ISBN 978-3-540-33991-5.

[27] A. D. Marco, G. Chiola, and G. Ciaccio. Using a gigabit ethernet cluster as a distributed disk array with multiple fault tolerance. In *LCN '03: Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks*, page 605, Washington, DC, USA, 2003. IEEE Computer Society.

[28] M. Mense and C. Scheideler. Spread: An adaptive scheme for redundant and fair storage in dynamic heterogeneous storage systems. In *19th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, San Francisco, California, USA, 20.-22. Febr.,, Jan. 2008.

[29] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distrib. Syst.*, 12(10):1094–1104, 2001.

[30] M. Mitzenmacher, A. Richa, and R. Sitaraman. *The power of two random choices: A survey of the techniques and results*. P. Pardalos, S. Rajasekaran, and J. Rolim, Eds. Kluwer, 2000.

[31] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[32] Windows PC accelerators. www.microsoft.com/whdc/system/sysperf/perfaccel.mspx, 2006. www.microsoft.com.

[33] M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 50–59, New York, NY, USA, 2003. ACM.

[34] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (raid). In *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pages 109–116, New York, NY, USA, 1988. ACM Press.

[35] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience*, 27(9):995–1012, September 1997.

[36] M. Raab and A. Steger. "balls into bins" — A simple and tight analysis. In *Lecture Notes in Computer Science*, volume 1518, pages 159–170, London, UK, 1998. Springer-Verlag.

[37] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Computer Communication Review*, volume 31, pages 161–172. Dept. of Elec. Eng. and Comp. Sci., University of California, Berkeley, 2001.

[38] C. Ruemmler and J. Wilkes. UNIX disk access patterns. In *Usenix Conference*, pages 405–420, Winter 1993.

[39] K. Salzwedel. *Data Distribution Algorithms for Storage Networks*. Dissertation, Universität Paderborn, Heinz Nixdorf Institut, Theoretische Informatik, 2004. EUR 20,00 ISBN 3-935433-62-X.

[40] C. Scheideler. *Probabilistic Methods for Coordination Problems*. HNI-Verlagsschriftenreihe 78, University of Paderborn, 2000. See also `http://www14.in.tum.de/personen/scheideler`.

[41] P. Scheuermann, G. Weikum, and P. Zabback. Data partitioning and load balancing in parallel disk systems. *VLDB Journal: Very Large Data Bases*, 7(1):48–66, 1998.

[42] C. Schindelhauer, S. Böttcher, and F. Rammig. The design of pamanet the paderborn mobile ad-hoc network. In *MobiWac '04: Proceedings of the second international workshop on Mobility management & wireless access protocols*, pages 119–121, New York, NY, USA, 2004. ACM.

[43] C. Schindelhauer and G. Schomaker. Weighted Distributed Hash Tables. In *SPAA '05: Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 218–227, New York, NY, USA, 2005. ACM.

[44] C. Schindelhauer and G. Schomaker. SAN optimal multi parameter access scheme. In *ICNICONSMCL '06: Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06)*, page 28, Washington, DC, USA, 2006. IEEE Computer Society.

[45] G. Schomaker. DHHT-RAID: A distributed heterogeneous scalable architecture for dynamic storage environments. In *AINA '07: Proceedings of the 21st International Conference on Advanced Networking and Applications*, pages 331–339, Washington, DC, USA, 2007. IEEE Computer Society.

[46] G. Schomaker. Deterministic unit ring decomposition and dhht. Technical report, Heinz Nixdorf Institut, September 2008.

[47] G. Schomaker, A. Brinkmann, F. Meyer auf der Heide, and U. Rückert. Verfahren zur verwaltung von metainformationen zur verteilung von datenblöcken über computerlesbare speichermedien sowie computerprogrammprodukt und computerlesbares speichermedium, October 2004.

[48] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In R. Guerin, editor, *Proceedings of the ACM SIGCOMM 2001 Conference (SIGCOMM-01)*, volume 31, 4 of *Computer Communication Review*, pages 149–160, New York, Aug. 27–31 2001. ACM Press.

[49] K. Talwar and U. Wieder. Balanced allocations: the weighted case. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 256–265, New York, NY, USA, 2007. ACM.

[50] U. Troppens and R. Erkens. *Speichernetze - Grundlagen und Einsatz von Fibre Channel SAN, NAS, iSCSI und InfiniBand*. dpunkt.verlag GmbH, Heidelberg, 2007.

[51] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[52] F. Wang. A case for redundant arrays of hybrid disks (rahd). In *INTERMAG '08: International Magnetics Conference*, Washington, DC, USA, 2008. IEEE.

[53] U. Wieder. Balanced allocations with heterogenous bins. In *SPAA '07: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 188–193, New York, NY, USA, 2007. ACM.

[54] R. Zimmermann and S. Ghandeharizadeh. HERA: Heterogeneous extension of RAID. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*. CSREA Press, June 2000.