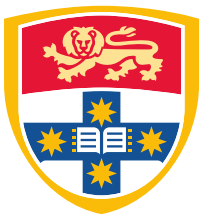

THE UNIVERSITY OF SYDNEY

Random Data Layouts

CAMERON SILVESTRINI

SID: 310 224 063

Supervisor: Dr. Julián Mestre

This thesis is submitted in partial fulfilment
of the requirements for the degree of
Bachelor of Science (Advanced) (Honours)

School of Information Technologies
The University of Sydney
Australia

November 5, 2013

STUDENT PLAGIARISM: COMPLIANCE STATEMENT

I certify that:

I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure;

I understand that failure to comply with the Student Plagiarism: Academic Board Policy: Academic Dishonesty and Plagiarism can lead to the University commencing proceedings against me for potential student misconduct under the 2012 Academic Dishonesty and Plagiarism in Coursework Policy.

This Work is substantially my own, and to the extent that any part of this Work is not my own I have indicated that it is not my own by Acknowledging the Source of that part or those parts of the Work.

Name: **Cameron Silvestrini**

Signature: **Date:**

ABSTRACT

Consider a data centre having a collection of m data items that must be stored on an array of n storage disks—each having a finite storage capacity k and a certain bandwidth—which needs to satisfy the clients' demand for the data items, e.g. consider movies to be streamed online: users request a movie from the service, and a copy of that movie is selected from one of the storage disks and then streamed to the user over the internet. We investigate an effective data layout algorithm to minimise the load across the disks to reduce congestion and ensure that the connection quality is not compromised for the users. We consider the case where demands are entirely unknown, and show that our uniform random data layout algorithm can still achieve very good performance.

We present two different aims that a data centre operator might have: the load balancing objective (minimise the maximum load across the servers), and the maximum coverage objective (maximise the total client demand that can be handled). In this thesis, we primarily consider the load balancing objective, and provide an analysis for the uniform random data layout that can be applied to any given distribution of demands. An algorithm is designed to compute a worst-case bound on the maximum load which holds true with very high probability. This algorithm was shown to be $\mathcal{O}(m^{t+1})$, where t is a parameter affecting the degree of accuracy of the analysis. We also show that for a specific but representative variety of demand distributions, our layout gives a 1.4-approximation of the optimum uniform data layout.

CONTENTS

LIST OF FIGURES

LIST OF PROGRAMS

CHAPTER 1

Introduction

Imagine a company has a large data centre, which streams movies or other multimedia to customers via the internet all over the world. Its storage needs are considerable, and the bandwidth requirements are immense: a large customer-base is in constant demand of the high-quality videos, music, etc., and the load on the servers is taxing. The items in the multimedia collection vastly outnumber the servers in the data centre, so a means of allocating those items among the storage disks is needed—one which spreads the customers' demand out evenly over the servers.

This is the canonical example of a data layout. The growing interest in cloud storage and computing shows the increasing applications of data layout theory to the problems faced by large data centres—particularly those with high bandwidth and storage requirements. A simple solution is to copy each data item onto every server. This gives the greatest scope to ensure the user demand is spread evenly over all servers, preventing any one server from becoming overloaded; however, our storage disks have limited capacity, and so if the items to store are large, we can only fit a few of them on each server. This will lower data redundancy; each data item can only be copied onto at most a few different servers, which increases the likelihood of one particular server becoming overloaded with demand.

The problem setup we will consider is as follows. Each movie in the collection is duplicated and stored on a number of different storage disks, which each have a finite (constant) capacity. Users request a movie from the service, and a copy of that movie is selected from one of the storage disks and then streamed to the user over the internet. The degree to which each movie is in demand will vary across the movies, and we will focus on the case where the demands are unknown ahead of time. We are interested in finding an effective data layout algorithm to minimise the load across the disks to reduce congestion and ensure that the connection quality is not compromised for the users.

The focus of this paper is on *uniform random data layouts*, where each item is allocated to a randomly-chosen constant number of servers. All previous work on data layout problems assume item demands are known ahead of time. We will concentrate on the situation of unknown demands, and will show that this data layout algorithm is effective for this problem.

1.1 Contributions

Our main contributions to this area of research are in the analysis of random data layouts. We introduce two different objectives for evaluating data layouts: maximum coverage and load balancing. In the *maximum coverage data layout problem* we are given client demands for each data item $d_1 \geq d_2 \geq ... \geq d_m$, and where we limit the amount of client demand that each disk can handle to some fixed load L. The objective is to maximise the amount of demand that the data layout can handle. In the *load balancing data layout problem*, we are given item demands $d_1, ..., d_m$, with the objective of minimising the expected maximum load on the disks. Our contributions will mainly apply to the load balancing objective, where we aim to present and analyse our random layout algorithm, and give an upper bound on the expected worst-case maximum load given a variety of demand distributions.

In chapter 2, we summarise the findings of related research in data layouts—in particular, focussing on the contributions relevant to random data layouts—before introducing our uniform data layout problem formally in chapter 3. Following this, we present the uniform random data layout algorithm—the main focus of this thesis—in which item copies are distributed across the storage disks uniformly at random.

Bounds for the worst-case maximum load of this algorithm are proved analytically in chapter 3. To do this, we initially follow the lead of Hoory, Linial and Wigderson [HLW06] in their proofs of properties of "magical graphs", and later tailor this approach to provide tighter bounds for our specific application of expander graphs.

In chapter 4 we analyse the performance of the uniform random data layout algorithm on different demand distributions, using the bounds proved in chapter 3. These bounds are further improved in chapter 5, with a tighter analysis using a novel approach of partitioning the item demands. We then demonstrate the improved estimates on the same theoretical demand distributions as used in chapter 4.

As a point of comparison for judging the performance of our algorithm, in chapter 6 we compute a tight bound on the performance that can be expected from any uniform data layout. We show that the uniform random algorithm achieves results that are near-optimal in many cases, with evidence suggesting that the algorithm gives a 1.4-approximation for the load balancing objective.

We begin to look at the alternative objective of maximum coverage in chapter 7, where we present some initial findings of the performance one can expect from the same algorithm.

CHAPTER 2

Background

In this chapter, we review the scientific literature related to data layouts, with a particular focus on random data layouts. We begin by looking at the maximum coverage problem, or as it is often referred to in the literature, *data placement*. This is followed by a discussion of *data migration*, which deals with changing demands over time. We then begin to detail other research in *random data layouts*, before elaborating on *expander graphs*, which inspired the initial analysis we conduct in chapter 3. Finally, we address *randomised load balancing*, which most closely relates to the focus of this thesis.

2.1 Data placement

The maximum coverage problem, one of the two possible objectives for the data layout problem outlined earlier, has been studied particularly extensively [Gol00; KK06]. The maximum coverage data layout problem is typically referred to in the literature as *data placement*. For instance, Golubchik et al. [Gol00] study this problem for two different classes of storage systems: *homogeneous* and *uniform ratio*. The problem formulation for homogeneous data placement case requires that all storage disks are identical, in that they have the same storage capacity k and support the same load L. The uniform ratio formulation allows the capacity C_j and load L_j of each storage disk j to vary, with the restriction that the ratio L_j/C_j remains constant across all disks.

In this paper, Golubchik et al. demonstrate the relationship between the data placement problem and the class-constrained multiple knapsack problem [SnT01; ST01], and show that the sliding-window algorithm presented by Shachnai and Tamir [SnT01] fits the bounds they prove for both the homogeneous and uniform ratio data placement problems. It is shown that it is always possible to pack a $(1 - 1/(1 + \sqrt{k})^2)$-fraction of items for any homogeneous data placement instance, under certain assumptions. They also strengthen the NP-hardness result from [SnT01] for the homogeneous case, and provide a polynomial time approximation scheme for both data placement problems.

The original sliding-window knapsack packing algorithm, on which Golubchik et al. based their data placement algorithm, was developed by Shachnai and Tamir [SnT01]. In their paper, they present results on two variants of the classic knapsack problem, where items of different types are to be placed in multiple knapsacks, which have finite capacity and bounds on the number of different types of items they can hold. The *class-constrained* multiple knapsack problem is analogous to the homogeneous data placement problem, with the objective of maximising the total number of packed items. They also consider the *fair placement problem*, where the goal is to place the same large portion of each set into knapsacks.

They prove that both variants are NP-hard in the general case, and develop approximation algorithms for both, with an approximation ratio dependent on the "uniformity" of the knapsacks. If we let the volume and number of compartments (the number of different types of items that are permitted) for each knapsack be V_j and C_j respectively for all knapsacks j, then given $r > 0$ and $\alpha \geq 1$ such that $r \leq V_j/C_j \leq \alpha \cdot r$ for all j, the moving-window procedure they provide achieves a $(1/\alpha)$-approximation for both problem variants.

The moving-window algorithm fills a knapsack K_j with items from a number of sets equal to its compartment size C_j. It moves a C_j-sized window along the sets of items in increasing order of cardinality until there are sufficiently many items to fill the knapsack (i.e. at least V_j items). This is the algorithm off which Golubchik et al. [Gol00] based both their homogeneous and uniform ratio data placement algorithms, additionally describing a new implementation improving the running time of the original from $\mathcal{O}(NM)$ to $\mathcal{O}((N+M)\log(N+M))$, where M and N are respectively the number of data items and the number of storage disks.

Kashyap and Khuller [KK06] focus on the homogeneous data placement problem, but unlike Golubchik et al. [Gol00] they drop the assumption of *unit size* for all data items by introducing a size parameter s_i for all items i. They develop two separate polynomial time approximation schemes for the case where $s_i \in \{1, ..., \Delta\}$, for some constant Δ; one for when the storage capacity k is constant, and one for arbitrary k. The algorithm for arbitrary k guarantees that a $((k-\Delta)/(k+\Delta))(1 - 1/(1+\sqrt{k/(2\Delta)})^2)$-fraction of items can be packed, under certain assumptions.

Furthermore, they prove tighter bounds for the case when the data items sizes are restricted to two sizes: small and large, i.e. $s_i \in \{1, 2\}$ for all i. The bound improves to a $(1 - 1/(1 + \sqrt{\lfloor k/2 \rfloor})^2)$-fraction of items being assigned.

Much more research about data placement, which we refer to as the maximum coverage data layout problem, has been conducted than for the load balancing data layout problem, which is to be the main focus of our research. However, the two problems are nonetheless closely related; load balancing is essentially a relaxation of one restriction of the maximum coverage problem formulation, and is a crude way of approximating the maximum coverage objective.

2.2 Data migration

A related problem is that of *data migration*, where data layouts are recomputed to adapt to changing demand. The problem is essentially to design an algorithm to organise the transfer of the data items copies stored on the disks in a series of rounds. This problem is shown by Khuller, Kim and Wan [KKW03] to be NP-hard. They present polynomial time approximation algorithms for the data migration problem, for which they prove a 9.5 approximation factor bound. This was later improved upon by Khuller, Kim and Malekian [KKM06] with a new $6.5+o(1)$ approximation algorithm. They present two different variations on this problem: *half-duplex*, where during each transfer round each disk can either send or receive a stored item, and *full-duplex*, where each disk can both send and receive during a round. Using external storage disks for high-demand items, they improve their algorithm further and achieve a 4.5 approximation factor for the half-duplex model. For the full-duplex model they develop an improved bound of $4 + o(1)$ without requiring the use of external disks.

Golubchik et al. [Gol06] also analysed the data migration problem, and adapted some of the algorithms from Khuller, Kim and Wan [KKW03], Hall et al. [Hal01] and Anderson et al. [And01]. They additionally identify a related sub-problem, the *correspondence problem*, whose solution was found to have a significant impact on the overall data migration problem. The correspondence problem involves the assumption that what is important is the grouping of the set of data items in each disk, and not the overall ordering of those items within the disk. They break the data migration process into two stages, the first step being to find a correspondence between the initial data layout and the target layout, and the second step being the classic data migration problem to minimise the number of rounds needed to achieve that target.

Establishing a correspondence involves constructing a weighted bipartite graph between the initial layout and the target layout, where an edge is added between two disks i and j' if both disks have the same capabilities (i.e. can store the same number of items, etc.), with a weight set to the number of new data items that need to be added to i to make it become j'. The objective then becomes to compute a minimum weight perfect matching in this graph, which gives a correspondence that minimises the total number of changes, but not the total number of rounds. This latter problem is solved by the data migration algorithms discussed above.

Golubchik et al. [Gol06] performed elaborate testing on their proposed algorithms. They tested a number of both correspondence and data migration algorithms to determine experimentally their practical performance, analysing their effectiveness against several different distributions of demand, such as the Zipf and Geometric distributions. They were able to develop algorithms that improved upon performance in practice, albeit at the expense of poorer worst-case behaviour.

Unlike in the works of Khuller, Kim and Wan [KKW03] and Golubchik et al. [Gol06], in the version of our problem we have unknown demands, so are not given any information regarding the demand of each item for the initial data layout. Thus another interesting problem to study

would be to adapt the data layout in a series of rounds, just like in the traditional data migration problem, and modify them to better fit the observed item demands.

2.3 Random data layout

In comparison with all of the deterministic data layout algorithms discussed so far, random data layouts were analysed by Santos, Muntz and Ribeiro-Neto [SMRN00]. Specifically, they discuss the practical advantages of a random data layout for multimedia storage servers and compare it with a traditional data striping technique. *Data striping*, discussed in [Ber94; ORS96; SV97], involves carefully splitting the data items and distributing them across disks, and is particularly advantageous for streaming applications where data access is typically highly sequential and predictable; however, Santos, Muntz and Ribeiro-Neto highlight several factors which in practice degrade data striping performance. Media compression techniques often produce variable bit rate (VBR) media streams, which decrease the predictability of data access. Further complications like multi-resolution encoding schemes found in the MPEG standards are discussed in [CK93].

They develop a random data layout multimedia storage server (which they call Randomized I/O, or RIO) and compare it to traditional data striping solutions for constant bit rate (CBR) traffic, to which data striping is better suited. They found that their random data layout solution was competitive with and often outperformed traditional striping schemes for CBR streams, and assert that the improvements would be even higher for VBR streams, to which striping is even less suited.

Santos, Muntz and Ribeiro-Neto also discuss the benefits of data replication, where a fraction of the data blocks randomly selected are replicated on other randomly selected disks, which allows the system some flexibility in selecting to which disk a request is routed. Mitzenmacher [Mit96] shows the theoretical improvement to expected maximum load when allowing a random process like this additional choices (further discussed in [ELZ86; RMS01]). Santos, Muntz and Ribeiro-Neto have exploited this for practical gains, causing an exponential reduction in the long tail for their response time distribution.

These results are promising for our application of random data layouts to uncertain demands, as we hope to utilise the advantages of data replication as shown by Richa, Mitzenmacher and Sitaraman [RMS01], and the benefits of the randomness of the data placement, to ensure that high-demand items are distributed evenly across the array of storage disks with high probability.

2.4 Expander graphs

Another set of properties we expect to use in our analytical proof of the worst-case maximum load performance bound of the uniform random data layout is that from expander graphs, the first explicit constructions of which were given in [Mar73]. A recent discussion of the

applications of expander graphs is given by Lubotzky [Lub12], and an example of their use in data stream algorithms is seen in the work of Ganguly [Gan08].

An expander graph is roughly defined as a graph where every "small" subset of vertices has a "large" boundary, for some definition of small and large. We are particularly interested in vertex expanders, where the boundary ∂S of a subset of vertices S is defined as the size of the set of neighbours of S. If we can construct a bipartite vertex expander graph such that every small subset of data items expands to a large subset of storage disks (where an edge from a data item to a storage disk indicates that the disk holds a copy of that item), then there will be sufficient data redundancy to ensure a low maximum load across the storage disks.

Hoory, Linial and Wigderson [HLW06] give a comprehensive overview of expander graphs, and we will aim to follow their proof of *magical graph* properties [HLW06, p. 447–8] and adapt it to our random data layouts to provide some initial loose bounds on the load balancing objective.

2.5 Randomised load balancing

There are some examples of previous research into load balancing using a random (as opposed to deterministic) approach. Sauerwald and Sun [SS12] analysed randomised load balancing in network topologies, where items (tokens) are reallocated in rounds to be evenly spread across processors. The continuous case, where tokens are infinitely divisible, corresponds to the well-known convergence of Markov chains. The authors focus on the discrete case which in many cases is more realistic, and were able to generalise the result from Friedrich and Sauerwald [FS09], which only applied to expander graphs, to all graphs.

Whereas previous works [Ber11; MGS98; RSW98] analysed rounding errors for the edges directly, Sauerwald and Sun present a novel technique that focuses on the tokens instead, and relate the movement of these tokens to independent random walks. This allows them to establish an analogy between the token distribution and the familiar and well-studied balls-and-bins model. Using this, they found that there is almost no difference between the continuous and discrete cases, with the two obtaining the (asymptotically) same load in the same number of rounds.

We aim to develop upon the previous research into random data layouts like this and those mentioned earlier, with a special focus into the case of uncertain demand, when item demands are not ahead of time and thus are not available to the data layout algorithm.

CHAPTER 3

Analysis

Having seen the related work and the state of the field, in this chapter we will formally define the uniform data layout problem, along with the two objectives: load balancing, and maximum coverage. We then introduce the uniform random data layout algorithm, and analyse its effectiveness with regard to specifically the load balancing objective.

3.1 Problem statement

Uniform data layout

Let $U = \{u_1, \ldots, u_m\}$ be a set of m data items, with corresponding demands $d_1, \ldots, d_m$, and let $V = \{v_1, \ldots, v_n\}$ be a set of n storage disks. Each disk has a constant capacity k of items it can store, and ℓ is the (fixed) number of copies to be made of each data item. We will denote the set of data item *copies* by U', a multiset containing each data item u_i for $i = 1, \ldots, m$ with multiplicity ℓ (so $|U'| = \ell m$). Similarly, let V' be the set representing *storage slots* on the servers (so V' is a multiset containing each v_j for $j = 1, \ldots, n$ with multiplicity k; $|V'| = kn$).

We require that every copy made of an item is stored *somewhere*, so $|U'|$ must equal $|V'|$, that is, $\ell m = kn$. We assume the item demands are ordered such that $d_1 \geq \cdots \geq d_m$, and assume that neither the item demands nor their respective ordering are known a priori by our data layout algorithm.

A *uniform data layout* is an assignment of data items to storage disks such that we make ℓ copies of each data item, and that no storage disk accepts more than k copies.

That is, a data layout is a bijective mapping between multisets $f : U' \to V'$. For notational convenience, we will instead denote the layout by the function $T : U \to \mathcal{P}(V)$, where $\mathcal{P}(V)$ denotes the power set of V, i.e. each data item $u \in U$ maps to a subset of storage disks $T(u) \subseteq V$. So the set of storage disks $T(u)$ to which $u \in U$ maps has size $1 \leq |T(u)| \leq \ell$. We abuse the notation slightly and define an "inverse" function $T^{-1} : V \to \mathcal{P}(U)$, and say that for each

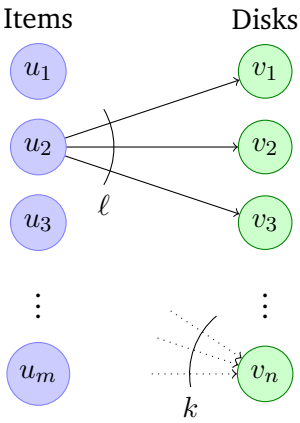

Figure 3.1: Visualisation of a data layout. The items and disks form a bipartite graph, where each item u_i has out-degree ℓ and each disk v_i has in-degree k.

disk $v \in V$, $T^{-1}(v)$ denotes the set of items that are stored in v. Hence $T : U \to \mathcal{P}(V)$ is a data layout if $1 \leq |T(u)| \leq \ell$ for all $u \in U$, and $|T^{-1}(v)| \leq k$ for all $v \in V$.

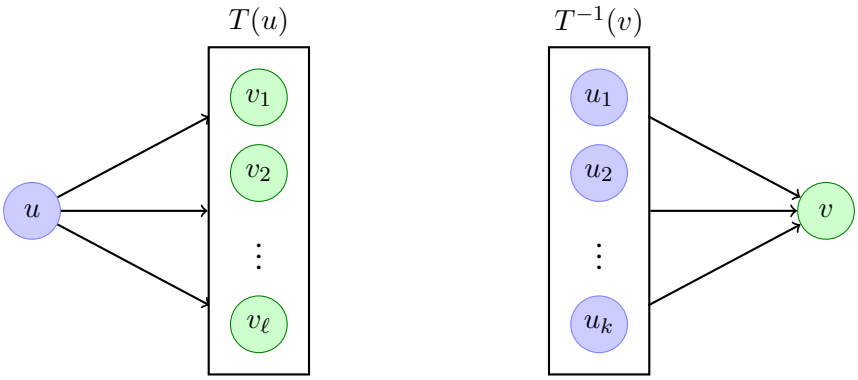

Figure 3.2: Illustration of the data layout function $T : U \to \mathcal{P}(V)$ and inverse function $T^{-1} : V \to \mathcal{P}(U)$.

The data layout T describes a way of distributing the incoming demand for the items across the servers. The demand for a given data item u can be split up between any or all of the servers in $T(u)$. Suppose that u_i is assigned to a subset $S = \left\{v_{j_1}, \ldots, v_{j_{|S|}}\right\} \subseteq V$ of servers (so $1 \leq |S| \leq \ell$). The demand d_i for item u_i can be divided between disks $v_{j_1}, \ldots, v_{j_{|S|}}$. The *load* on a disk is defined to be the sum of the assigned demand. We define the *maximum load* to be the highest load across all of the disks using the given data layout, assuming the demands are distributed optimally.

Let us now define the *average disk load* to be

$$\bar{L} = \frac{\sum_{j=1}^{m} d_j}{n}.$$

Load balancing objective

$\bar{L}$ represents the lowest possible maximum load across the disks, where the total demand $\sum_{j=1}^{m} d_j$ is spread evenly across all n disks. We will consider the maximum load as a multiple $\alpha \geq 1$ of $\bar{L}$.

The *load balancing objective* is to find a uniform data layout that minimises the maximum load α across the disks, while still serving the entire client demand.

Maximum coverage objective

The alternative objective, which is not discussed until chapter 7, is the *maximum coverage objective*. In this version of the problem, we place a strict limit on the load on each disk to be at most $\bar{L}$. The coverage is then the maximum portion of the client demand that can be successfully handled by the disks without any one of them exceeding this load. The aim is to find a uniform data layout to maximise the coverage.

For the rest of this chapter, and all chapters thereafter until chapter 7, we consider the load balancing objective exclusively.

3.2 Evaluating a layout

Network flow formulation

To evaluate a given a data layout $T : U \to \mathcal{P}(V)$ for the load balancing objective, we need a method of assigning the client demand to disks such that the maximum load is minimised. For this we can construct a network flow instance.

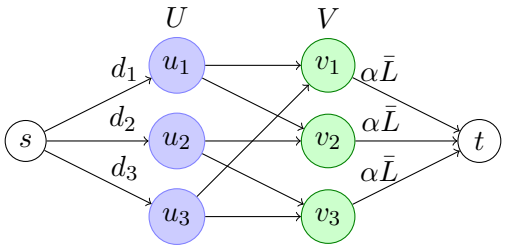

Figure 3.3: An example network flow instance for a data layout.

We begin by constructing a bipartite graph with the vertex set formed from the data items U and the storage disks V. For each item $u \in U$, we add an edge (u, v) with infinite capacity for every disk $v \in T(u)$ to which u is copied. We add two additional vertices s and t, connecting s to every vertex u_i with capacity equal to the demand d_i, and connecting each vertex v_j to t with capacity $\alpha\bar{L}$ for some constant $\alpha \geq 1$.

If the maximum flow saturates all edges out of s, then the data layout can satisfy all of the client demand $\sum_{j=1}^{m} d_j$ while keeping the maximum load to at most $\alpha\bar{L}$. However, if the maximum

flow is less than $\sum_{j=1}^{m} d_j$, then the maximum load must be more than $\alpha\bar{L}$. It is straightforward now to use binary search to find the smallest α^* that can ship $\sum_{j=1}^{m} d_j$ units of flow. In this case, we will say that α^* is the maximum load of the data layout.

Analytical formulation

For analytical purposes, it would be better to derive a less-algorithmic way of characterising the maximum load of a given data layout T. For a given subset S of items, let $d(S) = \sum_{u_i \in S} d_i$ be the total demand of items in S, and let $T(S) = \bigcup_{u_i \in S} T(u_i)$ be the set of storage disks to which the items in S are assigned. Then we define $L(S) = \bar{L} \cdot |T(S)|$, which reflects the "available load" for the items in S.

The total demand $d(S)$ for the item set S can only be assigned to the disks in $T(S)$. Hence the best maximum load possible for those storage disks occurs when the demand is spread evenly between them. Each will receive $\frac{d(S)}{|T(S)|}$ units of demand, which we can express as a multiple α of the average demand $\bar{L}$:

$$\frac{d(S)}{\bar{L} \cdot |T(S)|} = \frac{d(S)}{L(S)}.$$

Lemma 3.1. *The maximum load of a given data layout is given by*

$$\alpha^* = \max_{S \subseteq U} \frac{d(S)}{L(S)}.$$

Proof. We will prove this using the network flow instance described earlier (e.g. see Figure 3.3). The maximum load α^* was defined to be the minimum α such that the maximum flow in the network flow instance saturated the demand edges. That is, the maximum flow equals $\sum_{j=1}^{m} d_j$, or $d(U)$. We aim to show that $\max_{S \subseteq U} \frac{d(S)}{L(S)} = \alpha^*$, which we will do by proving that α^* is both an upper and lower bound for $\max_{S \subseteq U} \frac{d(S)}{L(S)}$.

Let $C = (R, \bar{R})$ be the minimum capacity $s-t$ cut in the graph. The edges in the cut-set of C must be of the form (s, u_i), $i = 1, \ldots, m$ with capacity d_i, or (v_j, t), $j = 1, \ldots, n$ with capacity $\alpha^*\bar{L}$. The only other edges in the graph are of the form (u_i, v_j) with capacity ∞, so clearly will not be in the mincut. Let S be the set of vertices in R that belong to U, then $S = R \cap U$. Then $T(S)$ must also be in R, since the edges in $E(S, T(S))$ have infinite capacity. There cannot be any vertices $v \in V \setminus T(S)$ included in R: doing so would mean paying the cost of the edge (v, t), which was $\alpha^*\bar{L} > 0$, hence moving v into $\bar{R}$ would lower the capacity of the cut. Therefore, $R = \{s\} \cup S \cup T(S)$.

Hence the capacity of the cut will be the sum of the edges from the disks in $T(S)$ to t, plus those from s to the items in $U \setminus S$. By the maxflow–mincut duality, the capacity of the mincut will

equal the maxflow $d(U)$, so:

$$\begin{aligned}
\alpha^* \bar{L}|T(S)| + d(U \setminus S) &= d(U) \\
\alpha^* \bar{L}|T(S)| &= d(U) - d(U \setminus S) \\
\alpha^* \bar{L}|T(S)| &= d(S) \\
\alpha^* &= \frac{d(S)}{\bar{L}|T(S)|} \\
\alpha^* &= \frac{d(S)}{L(S)}
\end{aligned}$$

We note that $\max_{S' \subseteq U} \frac{d(S')}{L(S')} \geq \frac{d(S)}{L(S)} = \alpha^*$, giving the "$\geq$" direction of the proof.

For the "$\leq$" direction, we note that if C is the minimum capacity $s{-}t$ cut with value $d(U)$, then all other cuts have capacity at least $d(U)$. So choose any arbitrary $S \subseteq U$, and construct the $s{-}t$ cut $(R, \bar{R})$ where $R = \{s\} \cup S \cup T(S)$. We saw earlier that this cut has capacity $\alpha \bar{L}|T(S)| + d(U \setminus S)$, and therefore

$$\alpha^* \bar{L}|T(S)| + d(U \setminus S) \geq d(U) \implies \alpha^* \geq \frac{d(S)}{L(S)}.$$

This is true for all $S \subseteq U$, hence $\max_{S \subseteq U} \frac{d(S)}{L(S)} \leq \alpha$. This gives us the "$\leq$" direction of the proof, and we are done. □

So Lemma 3.1 gives us a way of analytically determining the maximum load for a given data layout. This gives us the means by which to evaluate our proposed data layouts.

3.3 Random data layouts

The previous work on data layout problems assumes the demands $d_1, \ldots, d_m$ are known in advance and are available to the data layout algorithm, so that the items of highest demand can be prioritised. For this problem, we are interested in designing a data layout that will perform well, regardless of the specific distribution of the demands.

A commonly used approach when dealing with this kind of uncertainty is to used a randomised strategy. We will primarily consider the *uniform random data layout*, where we make the same number of copies of every item and distribute them uniformly at random among the disks. Notice that this strategy is oblivious to demands. An interesting question to investigate is what kind of demand patterns can such a layout handle, while still achieving a low maximum load.

Formal definition

Recall that U' denotes the multiset consisting of copies made of the data items U, where each $u \in U$ is present with multiplicity ℓ. Similarly, V' is the multiset of storage spaces, where

each storage disk $v \in V$ is included with multiplicity equal to its capacity k. Also recall that $|U'| = |V'|$.

A uniform random data layout is a random matching between U' and V', where for each $u \in U$ we choose ℓ elements from V' uniformly at random, without replacement. Let $T : U \to \mathcal{P}(V)$ be such a layout. Then $T(u) \subseteq V$ denotes the subset of unique disks in V chosen for each item $u \in U$.

Let (U, V, E) be the bipartite graph induced by the layout T, where $(u, v) \in E$ if $v \in T(u)$, that is, if item u is stored in disk v. For a subset $S \subseteq U$ we denote the neighbours of S by $T(S)$.

Our eventual aim is to find a good estimate of the maximum load for the uniform random data layout. To achieve this, we first need to find a way of measuring the probability of a given maximum load occurring.

3.4 Prelude to the analysis

The remainder of this chapter is dedicated to analysing the uniform random data layout algorithm, in order to establish an upper bound on the performance we can expect on the load balancing objective. We will provide two different analyses. The first given in section 3.5 is based on the concept of expander graphs, where we provide a guarantee on the maximum load given a required level of expansion. This is followed by an alternate analysis in section 3.6 that proved to be more versatile.

What we are interested in accomplishing is some form of guarantee regarding the performance of the data layout algorithm with regard to the maximum load across the disks. We will work towards developing a solution for this that is *demand-assignment agnostic*. What is meant by this is that we do not specify how the distribution of client demand is assigned to the data items; the upper bound found should hold true for any assignment of demands to disks. Additionally, the first analysis was designed to be *demand-distribution agnostic*, where not only is no requirement made on how the demands are assigned to disks, but also no assumptions made about the distribution of demands itself. We remove this restriction in the second analysis, which allows us to fix the demand distribution to give a tighter bound.

The rationale behind this requirement is related to the motivations for our problem definition. The item demands were assumed to be unknown to our data layout algorithm, but, in practice, we often find that there is some general trend that is known about the demands. That is, we might know the approximate shape the demand distribution might take, without actually knowing which items are the most popular. For instance, if we consider again the hypothetical movie-streaming website, we might find that the users' interests in movies changed regularly over time with regard to the specific movies they want to watch (e.g. the latest blockbusters), but might also find that the level of interest across all movies has an approximately fixed distribution. Hence we would like a guarantee that holds true regardless of how the demands are assigned to items, which is what we will develop in the remainder of this chapter.

More specifically, we will use probabilistic techniques to bound—with high probability—the worst-case maximum load for all demand assignments. That is, the probability that the uniform random layout generated by our random procedure has a worst-case maximum load no more than the given bound is very high. Note that the probability is taken over the space of uniform random layouts, not over the possible demand assignments. Therefore, it is possible that we could be very unlucky and generate a layout that achieves performance worse than our bound, but the chance of this happening is very low.

Note on terminology Throughout the rest of this thesis, we will frequently refer to results derived from the methods described in this chapter. We will often refer to this high-probability bound on worst-case maximum load performance as the *maximum load estimation* or the *expected maximum load*. This somewhat of a misnomer; we are not referring to the expectation value of the maximum load. Instead, this is meant to indicate the value which we expect (with high probability) to upper bound the maximum load for all demand assignments.

3.5 Expander graph analysis

In this section, we will design a method that will allow us to bound the probability of achieving a given maximum load, regardless of the particular demand distribution. Our proof involves specifying a minimum level of *expansion*, which will increase the likelihood of attaining a small maximum load. The proof is heavily inspired by the proof of *magical graph* properties given in [HLW06].

We refer to the *expansion* of a subset $S \subseteq U$ as being the size of the neighbourhood $T(S) \subseteq V$ in the data layout. Having larger expansions gives us more scope to spread the demand out evenly, giving a lower maximum load. We begin by defining an *minimum expansion* ρ_i for each subset $S \subseteq U$ of size i, such that the chance of a uniform random layout T achieving such an expansion is high.

We will ignore for the moment how exactly these values are defined or how they are computed, and return to that shortly. For now, assume we have integer parameters ρ_i for each $i = 1, \ldots, m$, telling us the minimum expansion required for every subset of items of size i. That is, $|T(S)| \geq \rho_{|S|}$ for all $S \subseteq U$. We say that a layout T satisfying this criterion has *sufficient expansion*.

Let us consider the probability that a layout has sufficient expansion,

$$\Pr\left[\forall S \subseteq U : |T(S)| \geq \rho_{|S|}\right].$$

This can be written in terms of *insufficient* expansion, which we can bound by summing the individual probabilities for each i:

$$\begin{aligned}\Pr\left[\forall S \subseteq U : |T(S)| \geq \rho_{|S|}\right] &= 1 - \Pr\left[\exists S \subseteq U : |T(S)| < \rho_{|S|}\right] \\ &\geq 1 - \sum_{i=1}^{m} \Pr\left[\exists S \subseteq U : |S| = i \text{ and } |T(S)| < \rho_{|S|}\right] \qquad (3.1)\end{aligned}$$

Ideally we would like this probability sum to be at most a small constant, say $\frac{1}{2}$, so we now provide a concrete definition for the ρ_i values, which will give us such a bound.

Definition 3.2. For $i = 1, \ldots, m$, let ρ_i be the largest integer such that

$$\Pr\left[\, \exists S \subseteq U : |S| = i \text{ and } |T(S)| < \rho_i \,\right] \leq 4^{-i}.$$

Having defined ρ_i in this way, we can establish our lemma.

Lemma 3.3. *The probability that a uniform data layout T has sufficient expansion with regard to the expansion parameters ρ_i for $i = 1, \ldots, m$ is at least $\frac{1}{2}$.*

Proof. Following from (3.1), we can use Defn. 3.2 to bound the probability sum

$$\sum_{i=1}^{m} \Pr\left[\, \exists S \subseteq U : |S| = i \text{ and } |T(S)| < \rho_{|S|} \,\right] \leq \sum_{i=1}^{m} 4^{-i} \leq \frac{1}{2},$$

which gives us the required result,

$$\Pr\left[\, \forall S \subseteq U : |T(S)| \geq \rho_{|S|} \,\right] \geq 1 - \frac{1}{2} = \frac{1}{2}.$$

□

Now that we have established the probability of sufficient expansion, we will use it in finding the likelihood of attaining a given maximum load.

Lemma 3.4. *Let T be a uniform random layout, ρ_i be a sequence of expansion parameters, and α be the target maximum load. If T has sufficient expansion with regard to ρ_i, and if $\sum_{j=1}^{i} d_j \leq \alpha \bar{L} \rho_i$ for all $i = 1, \ldots, m$, then the maximum load is at most α.*

Proof. To attain a maximum load of α, we need

$$\max_{S \subseteq U} \frac{d(S)}{L(S)} \leq \alpha,$$

or equivalently,

$$\frac{d(S)}{L(S)} \leq \alpha \quad \forall S \subseteq U.$$

Let S be any subset of U. Recall that the sequence of item demands is non-increasing, so that $d_1 \geq \cdots \geq d_m$. Hence $d(S) \leq \sum_{j=1}^{|S|} d_j$. Since by assumption T has sufficient expansion, $|T(S)| \geq \rho_{|S|}$. Finally, by definition $L(S) = \bar{L} \cdot |T(S)|$. So this gives us

$$\frac{d(S)}{L(S)} = \frac{d(S)}{\bar{L} \cdot |T(S)|} \leq \frac{\sum_{j=1}^{|S|} d_j}{\bar{L} \rho_i}.$$

Therefore, if $\sum_{j=1}^{|S|} d_j \leq \alpha \bar{L} \rho_{|S|}$ then $\frac{d(S)}{L(S)} \leq \alpha$. Since this holds for any subset S, the maximum load can be no greater than α. □

Corollary 3.5. *For any uniform random layout, if* $\sum_{j=1}^{i} d_j \leq \alpha \bar{L} \rho_i$ *for all* $i = 1, \ldots, m$*, then the maximum load is at most* α *with probability at least* $\frac{1}{2}$*.*

Proof. Follows directly from Lemma 3.3 and Lemma 3.4. □

It is now left to compute the actual ρ_i values. While there is no closed form for ρ_i, the values can be estimated. Better estimates will allow us to guarantee lower maximum loads. The following lemma gives a way of estimating ρ_i.

Lemma 3.6. *For* $i = 1, \ldots, m$ *where* $\ell i \leq (\rho_i - 1)k$*,*

$$\Pr\left[\exists S \subseteq U : |S| = i \text{ and } |T(S)| < \rho_{|S|}\right] \leq \binom{m}{i}\binom{n}{\rho_i - 1}\prod_{j=0}^{\ell i-1}\frac{(\rho_i - 1)k - j}{nk - j}.$$

Proof. Let S be a subset of U with cardinality i, and let R be a subset of V with cardinality $\rho_i - 1$ (so that R is of size strictly less than the expansion ρ_i required for S). We make ℓi copies of all the items in S. For any given copy, the probability that it ends up in R is $\frac{|R|}{|V|} = \frac{\rho_i - 1}{n}$. So the probability that $T(S) \subseteq R$ is at most $\left(\frac{\rho_i - 1}{n}\right)^{\ell i}$.

In fact, we have a tighter bound than this. Recalling that each disk $v \in R$ has a finite capacity k, if we instead consider the probability without replacement that $T(S) \subseteq R$, then the probability that each subsequent item copy ends up in R decreases as R becomes gradually more full.

That is, instead of each copy landing in R with probability $\frac{|R|k}{|V|k} = \frac{|R|}{|V|}$, the probability depends on j, the number of copies which have already landed in R: $\frac{|R|k-j}{|V|k-j} = \frac{(\rho_i - 1)k - j}{nk - j}$. Hence the chance that $T(S) \subseteq R$ is at most

$$\prod_{j=0}^{\ell i-1}\frac{(\rho_i - 1)k - j}{nk - j}.$$

There are $\binom{m}{i}$ such sets S and $\binom{n}{\rho_i - 1}$ such sets R. The statement of the lemma follows from a straightforward union-bound argument. □

Using Lemma 3.6, we can find numerically an estimate for ρ_i. Figure 3.4 shows the result for a specific demand pattern (demands modelled with a Zipf distribution, which we will frequently use again later on). We will see in the next section how we can come up with an even tighter result.

3.6 Alternate analysis

The analysis we just saw made no assumptions about the distribution of demands or the assignment of those demands to the items. Hence its conclusions remain valid for any demand distribution, but at the cost of severe overestimation in the majority of cases. In this section we

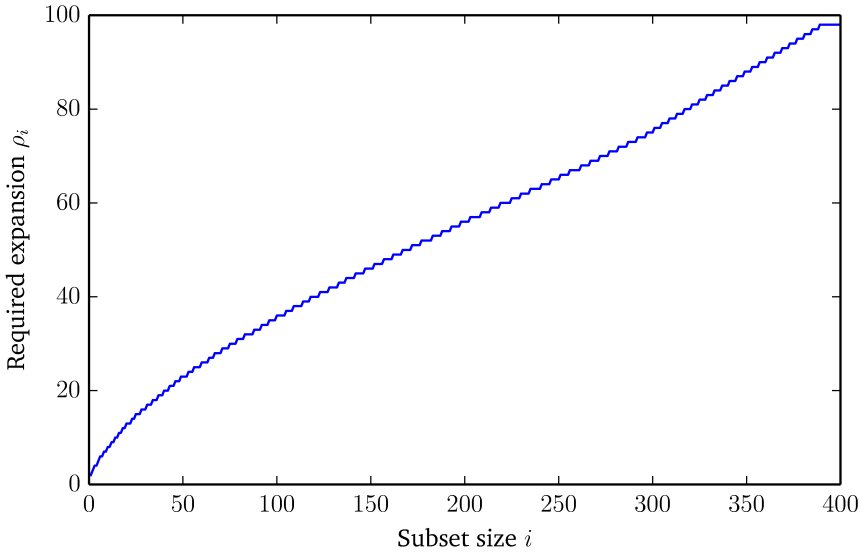

Figure 3.4: Required expansion ρ_i for subsets of size i. The demand distribution used was a Zipf distribution (characterised by parameters $s = 1$, $q = 0$; see section 4.1 for details), for a problem instance with $m = 400$ data items, $n = 100$ storage disks with capacity $k = 12$, and $\ell = 3$ copies made per item.

provide an alternative analysis, which begins by assuming the demand distribution $d_1, \ldots, d_m$ is fixed. This is not to say that we are now making the demands known to our data layout, but just that we assume the values are fixed for our analysis. We make no specific assumptions about the distribution itself, so the results will hold for *any* possible assignment of these demand values to the items. Furthermore, we are free to choose any demand distribution, and this procedure will still work.

So fix $d_1, \ldots, d_m$ to be an arbitrary demand distribution. That is, we know each value of the client demand, and have sorted them in descending order as before: $d_1 \geq \cdots \geq d_m$. However, while the demand values are known, the specific data item to which each corresponds is not known, as we wish this analysis to hold true for all such assignments.

The analysis will proceed in much the same way as our first analysis. Let us begin by defining the expansion parameters ρ_i for $i = 1, \ldots, m$ to be the minimum expansion required for subsets $S \subseteq U$ of size i to have a maximum load no larger than a given target α. That is, for a given $S \subseteq U$ with $|S| = i$, we have

$$\frac{d(S)}{L(S)} = \frac{d(S)}{\bar{L} \cdot |T(S)|} \leq \frac{\sum_{j=1}^{i} d_j}{\bar{L} \cdot |T(S)|},$$

which we want to be no larger than α. So,

$$\frac{\sum_{j=1}^{i} d_j}{\bar{L} \cdot |T(S)|} \leq \alpha \implies |T(S)| \geq \frac{\sum_{j=1}^{i} d_j}{\alpha \bar{L}}.$$

Hence we now have our definition for ρ_i:

Definition 3.7. For $i = 1, \ldots, m$,

$$\rho_i = \left\lceil \frac{\sum_{j=1}^{i} d_j}{\alpha \bar{L}} \right\rceil,$$

and a uniform random layout T is said to have *sufficient expansion* if $|T(S)| \geq \rho_{|S|}$ for all $S \subseteq U$.

Lemma 3.8. *Let T be a uniform random layout, let α be a target maximum load, and let ρ_i be expansion parameters defined with respect to the demand distribution $d_1, \ldots, d_m$ and α. If T has sufficient expansion, then the maximum load is at most α.*

Proof. The maximum load is given by $\max_{S \subseteq U} \frac{d(S)}{L(S)}$. Let S be a subset of items for which this maximum is attained, and let $i = |S|$. Then

$$\frac{d(S)}{L(S)} = \frac{d(S)}{\bar{L} \cdot |T(S)|} \leq \frac{\sum_{j=1}^{i} d_j}{\bar{L} \cdot |T(S)|} \leq \frac{\sum_{j=1}^{i} d_j}{\bar{L} \rho_i} \leq \alpha,$$

where the second inequality follows from the assumption that T has sufficient expansion (so $|T(S)| \geq \rho_i$), and the third inequality follows from the definition of ρ_i. Therefore the maximum load is at most α. □

For convenience, let us now define p_i for $i = 1, \ldots, m$ to be the probability that there is *insufficient* expansion for some subset $S \subseteq U$ of size i:

$$p_i = \Pr\left[\, \exists S \subseteq U : |S| = i \text{ and } |T(S)| < \rho_i \,\right].$$

Lemma 3.9. *For $i = 1, \ldots, m$ where $\ell i \leq (\rho_i - 1)k$,*

$$p_i \leq \binom{m}{i} \binom{n}{\rho_i - 1} \prod_{j=0}^{\ell i - 1} \frac{(\rho_i - 1)k - j}{nk - j}.$$

Proof. Proceeds in the same manner as the proof of Lemma 3.6. □

It is then easy to calculate an overall probability for a given maximum load α and demand distribution $d_1, \ldots, d_m$.

Lemma 3.10. *The probability that a uniform random data layout has maximum load more than α is bounded by the sum over all p_i. That is,*

$$\Pr\left[\max_{S \subseteq U} \frac{d(S)}{L(S)} > \alpha \right] \leq \sum_{i=1}^{m} p_i.$$

This allows us to estimate the probability numerically. The procedure is as follows. Given a demand distribution $d_1, \ldots, d_m$ and a target maximum load α, we compute the expansion parameters ρ_i using Defn. 3.7. From that, we can use Lemma 3.9 to estimate the p_i probabilities, which sum to give the overall probability that a uniform random data layout would exceed the target maximum load α.

By performing a binary search over values of α, we can find the smallest value α^* (to a certain level of precision) such that the maximum load across the disks is at most α^* with high probability. The allows us to estimate the worst-case maximum load for a uniform random data layout for any given demand distribution.

This procedure immediately suggests an algorithm for computing a worst-case maximum load bound of a uniform random data layout given a demand distribution. We have implemented such an algorithm, and in chapter 4 we perform a series of experiments on a wide array of demand distributions to evaluate its performance. First, however, we wish to specify a good baseline by which we can gauge such performance.

3.7 Baseline model

In order to evaluate the performance of the uniform random data layout algorithm, we will create a naïve baseline algorithm, which will be a deterministic layout that stores data item copies across the disks sequentially. This data layout also is also assumed to have no prior knowledge of the item demands, so cannot copy the items intelligently so that high-demand items share storage disks with low-demand items.

The algorithm proceeds as follows. Each data item is taken in turn, and ℓ copies are made and stored on ℓ disks. This is repeated for the first k items, so that the first ℓ storage disks are completely filled. We then move on to the next ℓ disks and the next k items (simplifying assumptions: ℓ divides n and k divides m).

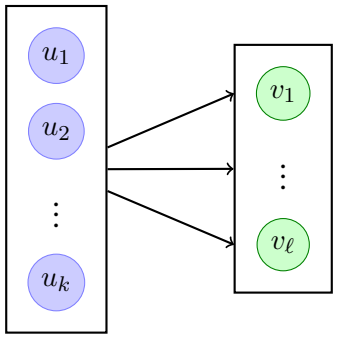

Figure 3.5: The deterministic baseline data layout. Each block of k items is copied onto a block of ℓ disks, with each disk holding one copy of each of the k items.

The maximum load possible with this approach occurs when the set S of the k items of highest demand get assigned to the same set of ℓ disks. By Lemma 3.1, the maximum load is

$$\frac{d(S)}{L(S)} = \frac{\sum_{j=1}^{k} d_j}{\bar{L}|T(S)|} = \frac{\sum_{j=1}^{k} d_j}{\bar{L}\ell}.$$

Although the chance of this exact event is very low, given any data layout of this kind and any demand distribution, there exists an assignment of demands to the data items such that this

maximum load can be obtained. By assigning the k highest demands to the same k items that are stored on a block of ℓ disks, we can obtain a maximum load of $\frac{\sum_{j=1}^{k} d_j}{\bar{L}\ell}$.

It is worth noting that this exact event is extremely unlikely to occur in practice, and so this baseline is very pessimistic. However, with high probability we can expect a similar assignment to occur (perhaps including only the first $k-1$ demands, say), which will be nearly this bad. Thus this serves as a good baseline by which to compare our uniform random data layout.

In the following chapter, we will evaluate the uniform random data layout algorithm on a range of example problem instances, using the maximum load estimation algorithm outlined earlier in this chapter. The above-mentioned deterministic data layout algorithm will serve as the baseline during those evaluations.

CHAPTER 4

Evaluation

In chapter 3, we established a procedure for bounding the worst-case maximum load of a random data layout, given a specific demand distribution $d_1, \ldots, d_m$. We also provided an appropriate baseline layout with which to compare the uniform random layout. In this chapter, we will evaluate the performance of the layout on a variety of different problem instances with different demand patterns.

The parameters governing the problem are the number of data items m, the number of storage disks n, and the capacity k of each disk. The number of copies ℓ made of each item is determined by the restriction that $\ell m = kn$. Finally, we have the demand distribution $d_1, \ldots, d_m$.

In this evaluation, we primarily consider problem instances with $m = 400$, $n = 100$, $k = 12$, and $\ell = 3$. We call this the *standard problem instance*. This is large enough for general trends to be observed, and yet still small enough for the computations to be completed in a relatively short time. In section 4.4, we consider additional problem sizes as evidence that the trends identified hold true in general.

4.1 The Zipf distribution

Following the lead of Golubchik et al. [Gol06], we will primarily consider demand distributions where the item demands are modelled according to Zipf's law, characterised by an exponent s. Each data item u_i has demand $d_i \propto 1/i^s$. The distribution is normalised to sum to 1, so in fact each demand is

$$d_i = \frac{1/i^s}{\sum_{j=1}^{m} 1/j^s}.$$

In [Gol06], it is noted that few large-scale studies exist for the kinds of applications we consider (e.g. movie streaming), so we instead consider purely theoretical distributions like Zipf that are potentially of interest.

A noteworthy characteristic of this distribution is for how quickly the values decrease; the first few items have extremely large demand, which quickly decreases to form a long tail. Changing

the exponent s allows us to vary the skewness of the distribution, giving us some control over how many "very popular" items we have. In [Gol06], the authors consider $s = 0.5$ and $s = 1$, which we will also use in our evaluations.

We will additionally consider *offsets* in the Zipf distribution. The Zipf distribution has a very strong positive skew, such that the first few values contain a very significant proportion of the probability mass. From a practical standpoint, it is highly unlikely that the demand for those highest-demand items could be completely unknown (e.g. consider new-release blockbuster movies: high demand is in no way unanticipated), so we could handle those select few with an alternative method tailored to their requirements, and use the uniform random layout for the rest. We can assume that such items have already been removed from our instance, and can model the new demand distribution with the tail of a Zipf distribution, which in practice is achieved by simply offsetting the indices in the Zipf generating function. So for an offset $q \geq 0$, $d_i \propto 1/(i+q)^s$. More specifically,

$$d_i = \frac{1/(i+q)^s}{\sum_{j=1}^{m} 1/(j+q)^s}.$$

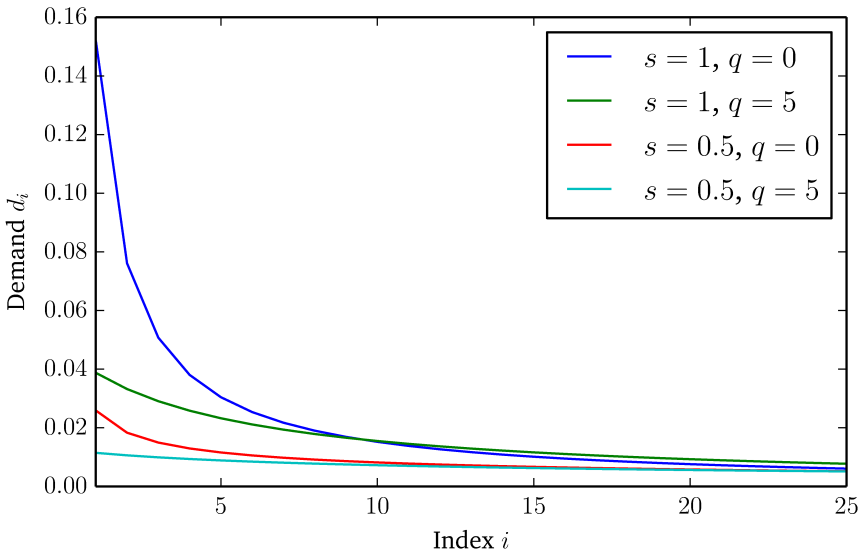

Figure 4.1: Zipf distributions, characterised by exponents $s = 1$ and $s = 0.5$, with offsets $q = 0$ and $q = 5$. Note how skewed the distribution is for larger s and small q.

4.2 Threshold phenomenon

Figure 4.2 plots the probability (calculated using Lemma 3.10) that the maximum load exceeds the target α, against a range of α values.

What is immediately evident from this graph is that the change in probability from 1 to near 0 occurs very suddenly and then tapers out. This justifies an approach of binary searching

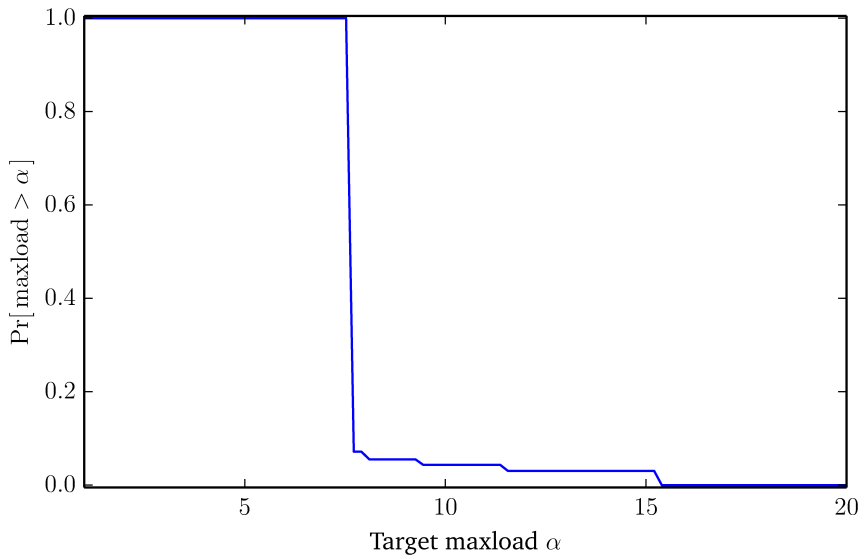

Figure 4.2: An upper bound on the probability estimates for the uniform random layout on the standard problem instance (defined at the start of chapter 4) illustrating the threshold phenomenon. The y-axis gives the probability that the maximum load is larger than some target value α, which is varied on the x-axis.

for our maximum load estimates. Instead of finding the point of near-zero probability (which would occur at about $\alpha = 16$ in Figure 4.2), we will instead find this sharp probability spike (at $\alpha = 7.5$). This spike drops to a fairly low probability (about 0.1), but as justification for this practice realise that we have been overestimating these probabilities, so measuring this sudden change provides the best estimate of the maximum load.

So our procedure going forwards will be to perform a binary search on α, to find at which target maximum load α the threshold between very high (≈ 1) and very low (≈ 0) occurs. We will let α^* denote this value in all future results. The level of precision used was set to 0.05 (that is, the reported α^* equals the true value to within ± 0.05), while binary searching for probability 0.5—in practice, this is effectively a search for the large spike shown in Figure 4.2.

This value α^* represents our estimate for the maximum load of a uniform random data layout for the given problem instance.

4.3 Results

We will now compare the uniform random layout against the baseline described in section 3.7. We first consider the standard problem instance, using a range of values for the Zipf offset q (representing increasingly uniform distributions as q increases), shown in Figure 4.3

We can clearly see that the uniform random layout gives a vast improvement on the baseline, especially for less-skewed demand distributions (which occurs for higher offsets q). If we adjust

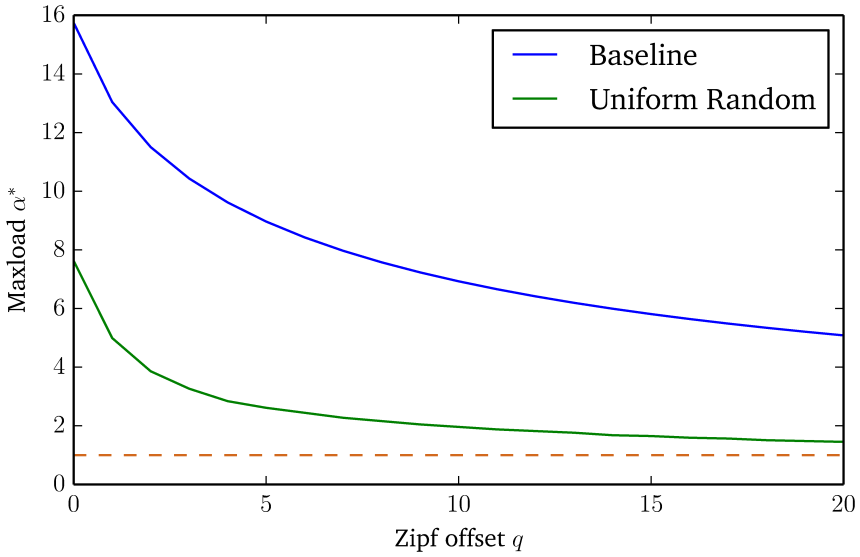

Figure 4.3: Comparison of the uniform random and the baseline deterministic data layouts on a Zipf distribution ($s = 1$) for a range of offsets q on the standard problem instance. The best possible maximum load is $\alpha^* = 1$, indicated by the dashed orange line.

the Zipf exponent parameter to $s = 0.5$, and thus reduce the skewness of the demand distribution further, we find that the estimate lowers to $\alpha^* \approx 1$ for all $q \geq 6$, as shown in Figure 4.4.

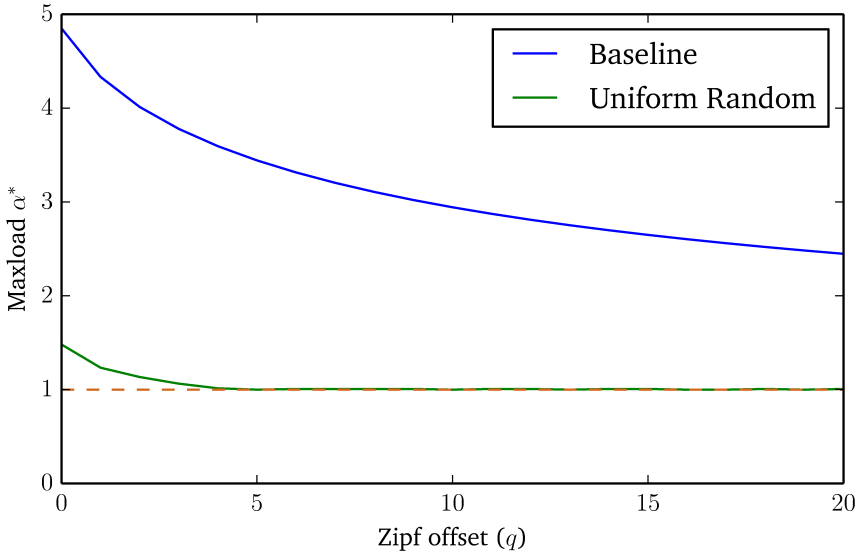

Figure 4.4: Comparison of the uniform random and the baseline deterministic data layouts on a Zipf distribution ($s = 0.5$) for a range of offsets q on the standard problem instance.

These results clearly illustrate how much better the uniform random data layout performs over the baseline. For a very steep Zipf demand distribution ($s = 1$, $q = 0$), we can see that the uniform random layout performs about twice as well as the baseline (attaining a maximum

load of less than 8 as compared to the baseline's maximum load of 16), and this improves further with less skewed distributions. For instance, when $s = 0,5$, $q = 0$, the uniform random layout achieves a maximum load of 1.5 compared with the baseline of almost 5.0. This shows us that the "harder" problem is the one with the more skewed demand pattern, whereby the first few items make up a very large portion of the total demand. As such, we will exclusively concern ourselves with the $s = 1$ case in future.

4.4 Other sizes

In this section, we will evaluate the uniform random data layout against the deterministic baseline for different problem instance parameters, to verify that above-mentioned results do not only hold just for the standard problem instance. Recall that we defined the standard problem instance to be where we have $m = 400$ data items, $n = 100$ storage disks with capacity $k = 12$, with $\ell = 3$ copies made of each item. We saw in Figure 4.3 and Figure 4.4 that higher values for the Zipf exponent s are harder problems, so we will let $s = 1$ for the results presented in this section.

We will begin by investigating how the behaviour changes as we vary the number of copies ℓ made of each data item. We keep $m = 400$ and $n = 100$ unchanged, so to meet the restriction that $\ell m = kn$, we will set $k = 4\ell$.

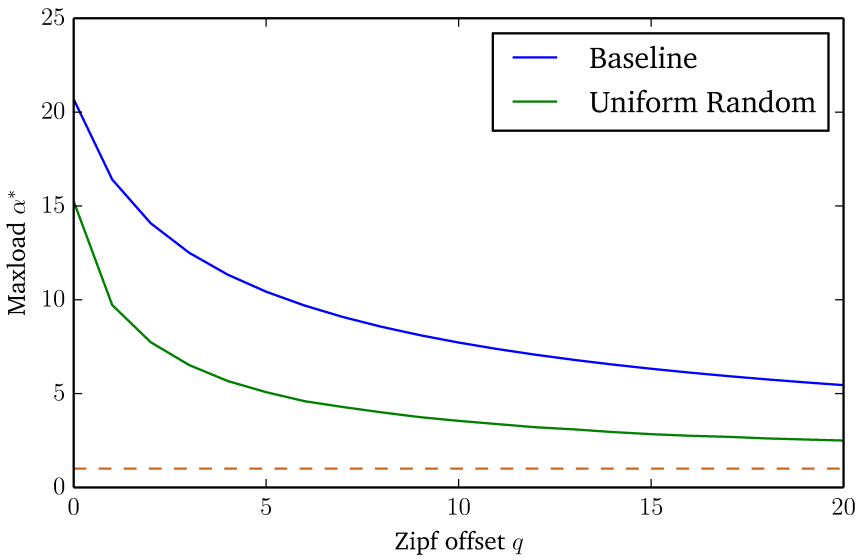

Figure 4.5: Maxload estimate comparison using only two copies of each item (standard problem instance with $\ell = 2$, $k = 8$).

Figure 4.5 presents the results when we only permit two copies to be made of each item. This restriction will mean that each data item is distributed to a fewer number of storage disks, and increases the chance that the demand for the most popular items is forced to be routed to a small set of storage disks, hence increasing the maximum load. This is particularly apparent

when $q = 0$: we see that the uniform random layout is only able to achieve a maximum load of 15 compared to the baseline's 20, whereas in Figure 4.3 where three copies are made we can halve the baseline's 16 to attain a maximum load of 8. Furthermore, even for large values of q we do not get very close to a maximum load of 1. We can see that the algorithm suffers from far worse performance when it is permitted to make only two copies of each item.

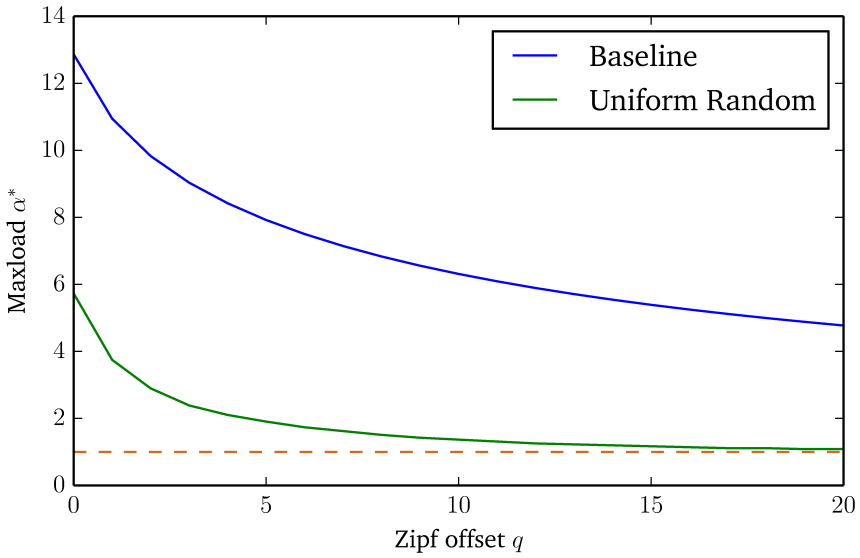

Figure 4.6: Maxload estimate comparison using four copies of each item (standard problem instance with $\ell = 4$, $k = 16$).

We can see that this trend continues in Figure 4.6, where four copies are made. The maximum load for both layouts improves further (to just over 12 for the baseline and just under 6 for the uniform random layout), since the items are spread out over more disks.

Figure 4.7 illustrates this trend for $q = 0$ using a range of values of ℓ. We can see that by making more copies, we improve the maximum load for both data layouts, with the only real anomaly being when $\ell = 2$. In that particular case, the random layout does not perform much better than the baseline. The level of redundancy achieved with only two copies is not high enough to provide any strong certainty that the client demand will be allocated evenly. Whereas for $\ell \geq 3$, we see that the uniform random layout performs about twice as well as the baseline.

Finally, we will consider a much larger problem instance. Let $m = 800$, $n = 200$, $k = 12$, and $\ell = 3$. This is essentially a scaled-up version of the problem used in Figure 4.3. Compare the results for this larger problem in Figure 4.8 with the equivalent smaller problem shown in Figure 4.3. While the maximum load has increased, the overall shape of the curves is unchanged, which help verify that the conclusions drawn in section 4.3 hold true in the general case.

We have seen that the uniform random data layout algorithm achieves very good performance compared to the baseline, for a wide variety of demand distributions. The performance gains tend to be much more significant when the demand distribution has less skew (this is seen in the

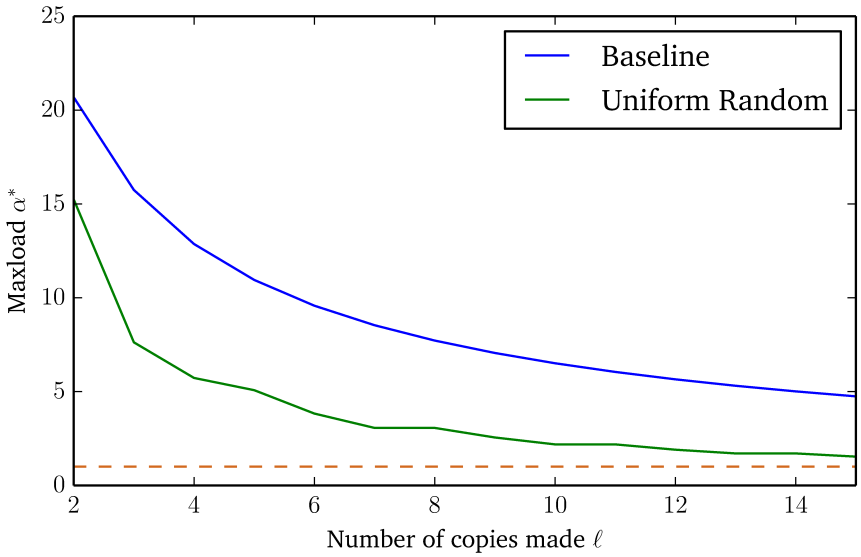

Figure 4.7: Maximum load estimate against the number of copies ℓ made of each item (standard problem instance with $m = 400$, $n = 100$, and $k = 4\ell$, using Zipf offset $q = 0$).

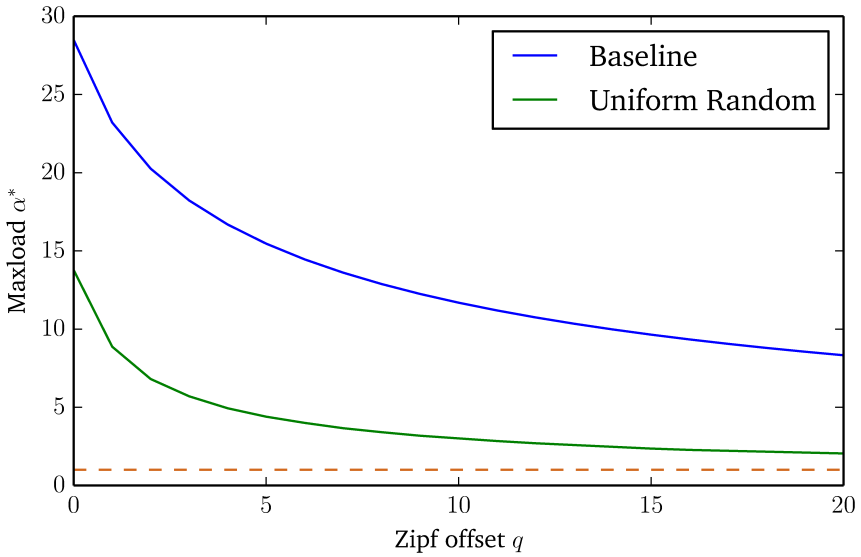

Figure 4.8: Maxload estimate comparison for a very large problem instance: $m = 800$, $n = 200$, $k = 12$, $\ell = 3$, with demands following the Zipf distribution with $s = 1$.

experiments conducted by the trials of higher Zipf offsets q and smaller Zipf exponents s). The reason for this is due to the first few items in the highly skewed distributions having such high demand. In chapter 5 we investigate potential strategies for handling these items to improve our analysis.

In this chapter, we also have seen how varying the number of copies made of each data items will affect the expected maximum load. We saw that in general, making additional copies helps lower the load, since we can spread the client demand for high-demand items over more disks, which is to be expected; however, it also became apparent that using only $\ell = 2$ copies per item is clearly insufficient. By allowing only one additional copy, we can halve the expected worst-case maximum load, as evidenced by Figure 4.7.

In the following chapter, we will see how we can further improve our analysis to attain even better results for the uniform random data layout.

CHAPTER 5

Partitioning the demand

We now present a more refined analysis for the load balancing objective where we consider the high-demand items separately.

The items with the highest demand individually contribute the most to the maximum load, and as we saw in chapter 4, this is particularly true for the heavily skewed Zipf distributions. As such, we would ideally like to ensure that these first few items have a large expansion, which would potentially allow us to reduce the upper bound on the worst-case maximum load.

We will begin the analysis by treating these high-demand items separately. This broadly consists of dividing the set of data items U in two, separating out the items with the highest demand. This treatment will lead towards tighter bounds for the maximum load estimate, and we will see how it can generalise to any number of partitions.

5.1 Analysis: two parts

Let us choose a threshold value $\theta \in [0, 1]$, which represents the percentage of items that we consider to be "in high demand". We will partition the set of data items U into two parts with approximate sizes θM and $(1 - \theta)M$. Let $\gamma = \lfloor \theta M \rfloor$ be the index of the last high-demand item, so let $U_H = \{1, \ldots, \gamma\}$ be the set of *high-demand items* and $U_L = \{\gamma + 1, \ldots, M\}$ be the set of *low-demand items*, such that (U_H, U_L) partitions U.

We can now define a new set of values $\rho_{i,j}$ for $i = 1, \ldots, \gamma$, $j = \gamma + 1, \ldots, m$, analogous to our previous ρ_i from Defn. 3.7. Define $\rho_{i,j}$ to be the minimum amount of expansion required to ensure a maximum load of at most α for the i most popular high-demand items and the j most popular low-demand items.

Definition 5.1. Given a partitioning index γ, for all $i = 1, \ldots, \gamma$, $j = \gamma + 1, \ldots, m$,

$$\rho_{i,j} = \left\lceil \frac{\sum_{h=1}^{i} d_h + \sum_{h=1}^{j} d_{\gamma+h}}{\alpha \bar{L}} \right\rceil .$$

A uniform random layout T is said to have *sufficient expansion* if $|T(S_H \cup S_L)| \geq \rho_{|S_H|,|S_L|}$ for all $S_H \subseteq U_H$, $S_L \subseteq U_L$.

We also define an analogous $p_{i,j}$ to be the probability of *insufficient expansion* for some subsets $S_H \subseteq U_H$, $S_L \subseteq U_L$ with $|S_H| = i$, $|S_L| = j$:

$$p_{i,j} = \Pr\left[\exists (S_H, S_L) \subseteq (U_H, U_L) : |S_H| = i, |S_L| = j \text{ and } |T(S_H \cup S_L)| < \rho_{i,j}\right].$$

The $\rho_{i,j}$ and $p_{i,j}$ values can be computed as before, using a corresponding set of upper bounds (proof is straightforward and proceeds in the same manner as for Lemma 3.6):

Lemma 5.2. *For all $i = 1, \ldots, \gamma$, $j = \gamma + 1, \ldots, m$ where $\ell(i + j) \leq (\rho_{i,j} - 1)k$,*

$$p_{i,j} \leq \binom{|U_H|}{i}\binom{|U_L|}{j}\binom{n}{\rho_{i,j} - 1}\prod_{h=0}^{\ell(i+j)-1}\frac{(\rho_{i,j} - 1)k - h}{nk - h}.$$

Lemma 5.3. *The probability that a uniform random data layout has maximum load more than α is bounded by the sum over all $p_{i,j}$. That is,*

$$\Pr\left[\max_{S \subseteq U}\frac{d(S)}{L(S)} > \alpha\right] \leq \sum_{i=1}^{\gamma}\sum_{j=\gamma+1}^{m} p_{i,j}.$$

By means of Lemma 5.3, we now have a new way of estimating the probability that a given maximum load cannot be satisfied. Just as we did in section 4.2, we can perform a binary search to find our best maximum load estimate. However, the question remains, exactly how do we choose the point at which we split the demands into two parts?

Note that this split point and the whole concept of partitioning the demand is for the purposes of our analysis only. It does not correspond to any real division of the data items or clients with regard to the actual data layout. This merely serves to help give a more accurate estimate of the maximum load for such a uniform random data layout.

5.2 Partitioning heuristics: two parts

Figure 5.1 shows the maximum loads than are obtained when the split point γ used to partition the item demands is changed. The rationale behind splitting the demand into multiple parts was to separate the very high demand items out, to improve the bounds found in our analysis. As one could expect from this, the optimum result is attained when only a few of the highest demand items are included in their own part. There is a sharp initial decrease in the maximum load as we increase γ, which is followed by a more gradual increase.

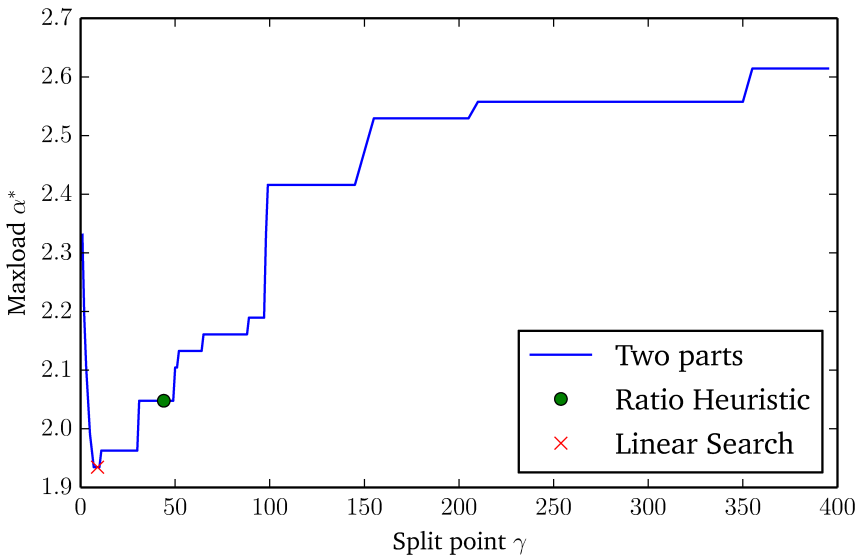

Figure 5.1: Maximum load estimate using two parts against split point. Uses the standard problem instance with Zipf offset $q = 5$. The points found using the ratio heuristic and linear search are also plotted.

Ratio heuristic

In order to choose the optimum split point for partitioning the data items, we first propose a simple heuristic which partitions the items such that the ratio between the highest and lowest demands in each part is minimised.

For each part, we consider the ratio of the highest demand to the lowest demand. So splitting the data items at γ gives us the ratios $\frac{d_1}{d_\gamma}$ and $\frac{d_{\gamma+1}}{d_m}$. We aim to choose γ such that the maximum ratio across the parts is minimised. That is,

$$\min_{\gamma} \left\{ \max \left\{ \frac{d_1}{d_\gamma}, \frac{d_{\gamma+1}}{d_m} \right\} \right\}.$$

The idea behind this heuristic is that we find a good compromise between the two parts. We do not want either part containing too many or too few items, so this should adjust γ to find a good trade-off. Furthermore, it can be easily extended to use three or more parts, by minimising the maximum ratio across *all* parts.

The ratio heuristic is also shown in Figure 5.1. We can see that, while it does produce a fairly good result, it is not optimal. Indeed, for other (heavily skewed) demand distributions, it chooses very poor split points.

Linear search

The very sharp initial decrease noticeable in Figure 5.1 turns out to be common for variations of the standard Zipf problem instance, with generally the optimum γ being very small. This

suggests an alternative approach.

Instead of using a heuristic to guess the optimum value, we can instead perform a linear search beginning at $\gamma = 1$, and look for a local minimum. Seeing as the optimum split point is often very small (in Figure 5.1 it occurs at $\gamma = 9$), this is not necessarily as inefficient as it might first seem. Furthermore, the time required to compute the maximum load estimate for each γ is very short for small γ. We must estimate the probability $p_{i,j}$ using Lemma 5.2, for all $i = 1, \ldots, \gamma$, $j = \gamma + 1, \ldots, m$. That is, the number of probability estimates is proportional to $|U_H||U_L| = \gamma(m - \gamma)$, which is smallest when γ is near 1 (or m). So the linear search should only involve a small number of fast-to-compute probability estimates.

The result for the linear search is shown Figure 5.1 as well. We can see that it correctly finds the optimum split point at $\gamma = 9$ with a maximum load of just over 1.9. This is the technique we will use to find the optimum split point in our uniform random layout evaluation using two parts.

5.3 Evaluation: two parts

The results on the standard problem instance using two parts is shown in Figure 5.2. We can see that, particularly for lower values of q (i.e. more heavily skewed demand distributions), partitioning the data items helps to improve the analysis considerably. When $q = 0$, we can see that a maximum load of approximately 5 is attained by using two parts, compared with nearly 8 for one part and 16 for the baseline.

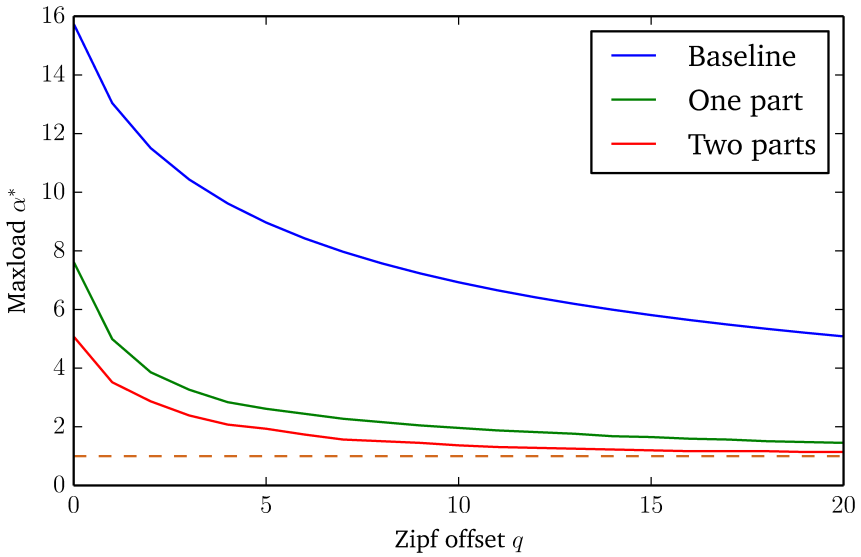

Figure 5.2: Maximum load estimate using two parts, compared with the one-part analysis and the baseline, for the standard problem instance.

Hence our new analysis with partitioned item demands leads to a considerable improvement in

our maximum load estimate. Figure 5.2 also illustrates just how well the uniform random data layout performs.

5.4 Analysis: t parts

Note that there is no reason necessitating only using two parts for the analysis. We may find that by partitioning the items further we can improve our estimate even more.

Let us define a partition of the items consisting of t parts $A_1, A_2, \ldots, A_t$ so that the demand for each item in A_i is larger than the demand for each item in A_j, for all $j > i$. That is, A_1 is the set of items $\{u_1, \ldots, u_{|A_1|}\}$, $A_2 = \{u_{|A_1|+1}, \ldots, u_{|A_1|+|A_2|}\}$, and so on.

Alternatively, we can say $A_r = \{u_{a_r+j}\}_{j=1,\ldots,|A_r|}$, where a_r offsets the start of that part:

$$\begin{aligned} a_1 &:= 0 \\ a_r &:= a_{r-1} + |A_{r-1}| \qquad r = 2, \ldots, t \end{aligned}$$

Let us now introduce some new notation for convenience. We let A_r^i denote the subset of A_r consisting of the first i items in A_r. That is,

$$A_r^i = \{u_{a_r+j}\}_{j=1,\ldots,i}$$

for $i = 1, \ldots, |A_r|$. We will let $A_r^0 = \emptyset$.

Now we can denote the sum of the first i demands in a part A_r by $d(A_r^i)$, where $d(\emptyset) := 0$.

Using this notation, we can define expansion parameters for our partitioned items analogous to our original ρ_i defined previously. Let $i_1, \ldots, i_t$ denote the number of included elements from each set $A_1, \ldots, A_t$. Then

$$\rho_{i_1,\ldots,i_t} = \left\lceil \frac{\sum_{r=1}^{t} d(A_r^{i_r})}{\alpha \bar{L}} \right\rceil,$$

where the index i_r for partition A_r can range over all or none of the items in the partition, so $i_r = 0, \ldots, |A_r|$ for $r = 1, \ldots, t$.

We again define our probability $p_{i_1,\ldots,i_t}$ in terms of $\rho_{i_1,\ldots,i_t}$:

$$p_{i_1,i_2,\ldots,i_t} = \Pr\left[\, \exists S \subseteq U : |S \cap A_r| = i_r \text{ for all } r \text{ and } |T(S)| < \rho_{i_1,\ldots,i_t} \,\right].$$

This gives us a means of estimating the maximum load computationally using multiple parts.

Lemma 5.4. *Let $i_1, \ldots, i_t$ be a sequence of indices for partitions $A_1, \ldots, A_t$, and let $I = i_1 + \cdots + i_t$. If $\ell I \leq (\rho_{i_1,\ldots,i_t} - 1)k$ then*

$$p_{i_1,\ldots,i_t} \leq \binom{|A_1|}{i_1}\binom{|A_2|}{i_2}\cdots\binom{|A_t|}{i_t}\binom{n}{\rho_{i_1,\ldots,i_t}-1}\prod_{j=0}^{\ell I-1}\frac{(\rho_{i_1,\ldots,i_t}-1)k-j}{nk-j}.$$

Lemma 5.5. *The probability that a uniform random data layout has maximum load more than α is bounded by the sum over all $p_{i_1,\ldots,i_t}$. That is,*

$$\Pr\left[\max_{S\subseteq U}\frac{d(S)}{L(S)} > \alpha\right] \leq \sum_{i_1,\ldots,i_t} p_{i_1,\ldots,i_t}.$$

5.5 Partitioning heuristics: t parts

Now that we have extended our analysis to use arbitrarily many parts when partitioning the item demand, we need to revise the heuristic by which we choose the sizes of the parts. In section 5.2 we presented two techniques for choosing the split point when using only two parts: the ratio heuristic, and the linear search. In this section we will adapt these to handle t parts.

Search space First let us consider the problem of choosing a good partition $A_1, \ldots, A_t$, assuming we have fixed the number of parts t. That is, we want to choose $t-1$ *dividers* among our m data items such that the t parts formed are non-empty. So we have $m-1$ positions in which to place the $t-1$ dividers, giving us $\binom{m-1}{t-1}$ different partitions (this is just the number of compositions of m into exactly t parts).

If t is considered fixed, then

$$\binom{m-1}{t-1} = \mathcal{O}(m^{t-1}),$$

giving us a fairly large search space. So while for $t=1$ it was possible to go without any form of heuristic and just check every possible split point γ, for larger t that is no longer viable, necessitating the use of some other heuristic.

Time complexity Assume now that we have chosen a partition $A_1, \ldots, A_t$ and wish now wish to compute the maximum load. The algorithm suggested by Lemma 5.5 involves summing each probability $p_{i_1,\ldots,i_t}$. So we will need to sum $|A_1| \times \cdots \times |A_t|$ different $p_{i_1,\ldots,i_t}$ values.

Note that the product $|A_1| \times \cdots \times |A_t|$ is maximised when all t parts are the same size. That is, the m items are evenly split between each part: $|A_1| = \cdots = |A_t| = \frac{m}{t}$.

If we assume for the moment that we can compute the upper bound from Lemma 5.4 in constant time, then we can calculate the overall time complexity:

$$\begin{aligned}\mathcal{O}(|A_1| \times \cdots \times |A_t|) &= \mathcal{O}\left(\left(\frac{m}{t}\right)^t\right)\\ &= \mathcal{O}(m^t)\end{aligned}$$

where the second equality follows from the assumption that t is fixed.

Let us now revisit the assumption that each individual probability estimate takes constant time. In fact, in order to compute each probability $p_{i_1,\ldots,i_t}$ (see Lemma 5.4), we must calculate the value of some binomial coefficients, and furthermore take the product of ℓI fractions (where

$I = i_1 + \cdots + i_t$). As a basic optimisation, we can pre-compute the binomial coefficients for the sizes of every part in the partition. That is, store the values $\binom{|A_r|}{j}$ for all $r = 1, \ldots, t$ and $j = 1, \ldots, |A_r|$, and for $\binom{n}{j}$ for all $j = 1, \ldots, n$. This would make those operations constant time. In the worst case we could have $I = i_1 + \cdots + i_t = |A_1| + \cdots + |A_t| = m$, which means computing

$$\prod_{j=0}^{\ell I - 1} \frac{(\rho_{i_1,\ldots,i_t} - 1)k - j}{nk - j}$$

is $\mathcal{O}(m)$. Hence the actual time complexity for computing the maximum load bound for a single partition is

$$\mathcal{O}(m^t \cdot m) = \mathcal{O}(m^{t+1}).$$

Note however that for many partitions the time taken to compute $p_{i_1,\ldots,i_t}$ will not nearly be this bad. In particular, we are mostly interested in the partitions where the first $t-1$ parts each hold only a few items, with the last part t holds the remainder of the long tail of the Zipf distribution, giving us far fewer values to compute.

What this does illustrate, however, is that a brute force search through all partitions quickly becomes intractable for more than two parts. Hence we will now consider adapting the partitioning heuristics presented earlier for arbitrary t.

Ratio heuristic

As discussed in section 5.2, for two parts this involves choosing the split point γ to minimise the maximum demand ratio. This is easy enough to find when there are only two parts, as we only need to check $m - 1$ possible values of γ; however, as we just saw, for larger t the search space is too big to explore completely, so we will need to modify this heuristic further.

Let the *demand ratio* for a given part A_r be the ratio of the highest demand to the lowest demand in that part:

$$\frac{\max d(A_r)}{\min d(A_r)}.$$

Since we have labelled the items in descending order by their demands, we know that $\max d(A_r)$ is the demand of the first item in A_r, and similarly $\min d(A_r)$ is the demand of the last item. If we introduce some new notation and let d_i^r be the demand of the ith item in partition A_r, i.e.

$$d_i^r = d_{|A_1|+\cdots+|A_{r-1}|+i},$$

then we have

$$\frac{\max d(A_r)}{\min d(A_r)} = \frac{d_1^r}{d_{|A_r|}^r}.$$

We will judge a partition by the maximum demand ratio across each of its parts. What we would like is to minimise the maximum ratio across all partitions:

$$\min_{A_1,\ldots,A_t} \left\{ \max_{r=1,\ldots,t} \frac{d_1^r}{d_{|A_r|}^r} \right\}.$$

So far this is identical to the two-part version defined in section 5.2; however, whereas before there were only $m-1$ different split points to check, for t parts we have $\mathcal{O}(m^{t-1})$ different possible partitions. Note that this is still much faster than actually computing the maximum load for each partition, since we can compute

$$\max_{r=1,\dots,t} \frac{d_1^r}{d_{|A_r|}^r}$$

in $\mathcal{O}(t)$ time, whereas finding the maximum load is $\mathcal{O}(m^{t+1})$.

Instead of checking every possible partition to find the smallest ratio, we will work the other way and try to find a partition that satisfies a certain ratio. That is, given a fixed demand ratio $R > 1$, we wish to find any partition $A_1, \dots, A_t$ satisfying

$$\max_{r=1,\dots,t} \frac{d_1^r}{d_{|A_r|}^r} \leq R.$$

We begin by letting $A_1 = \{u_1\}$ (so it has ratio $1 < R$). We progressively try to add the next data item to the current part (that is, add $u_j = u_2$ to part $A_r = A_1$), so long as doing so keeps the demand ratio for A_r below R. Finally, we will reach a point where adding u_j to A_r will cause the ratio to exceed R. We are then forced to create a new part: $A_{r+1} = \{u_j\}$, and continue from there.

Finally, we will have constructed a partition such that each part has demand ratio at most R. If the number of parts is more than t, then the target ratio R was too ambitious, and no such partition into t parts exists. If there are exactly t parts then the partition $A_1, \dots, A_t$ can be used. If there are less than t parts, it is trivial to move a few data items into their own part to find a partition with t parts meeting the target ratio R.

Hence now we have a means of quickly finding a partition with a low demand ratio. We can perform a binary search by adjusting R to find (to a given level of precision) the smallest ratio R^* and its corresponding partition.

Figure 5.3 shows the maximum load estimate using a partition of three parts chosen by this ratio heuristic, compared with a using a partition found by the linear search method, discussed next.

Linear search

We will now adapt the linear search from the version presented for two parts in section 5.2, and generalise it to t parts.

Given the number of parts t to use, we begin by initialising all but the last part to have size 1. That is, $A_1 = \{u_1\}, A_2 = \{u_2\}, \dots, A_{t-1} = \{u_{t-1}\}, A_t = \{u_t, \dots, u_m\}$. Then progressively consider each part, starting at A_{t-1} and working backwards to A_1. So beginning at part $A_i = A_{t-1}$, start with $|A_i| = 1$ and compute the maximum load for the partition $A_1, \dots, A_t$. Then

increment the size of A_i by 1 (taking the extra item from the last part A_t, that is, reducing $|A_t|$ by 1) and recompute the maximum load. This is repeated until doing so no longer improves the result. Then reduce i by 1 and continue.

There is a question of what to do when increasing the size A_i of partition i does not affect the maximum load. Ideally, we prefer to minimise the product of partition sizes (for faster computations), so we could stop when the maximum load has not strictly decreased. However, it is evident that there are "plateaus" in the maximum load plot, and it is possible that it could continue to decrease at some later point. Hence we should continue to explore these plateaus to some degree. Unfortunately the plateaus can continue for a very long time, far after the point where it becomes useful to check the maximum load. So a compromise is to introduce a "lookahead" parameter, whereby we explore plateaus for a short length (e.g. 10 increments in partition size) and look for future improvements. If nothing is found, return to the start of the plateau.

Results obtained using linear search through partitions having three parts is shown in Figure 5.3, compared with the ratio heuristic.

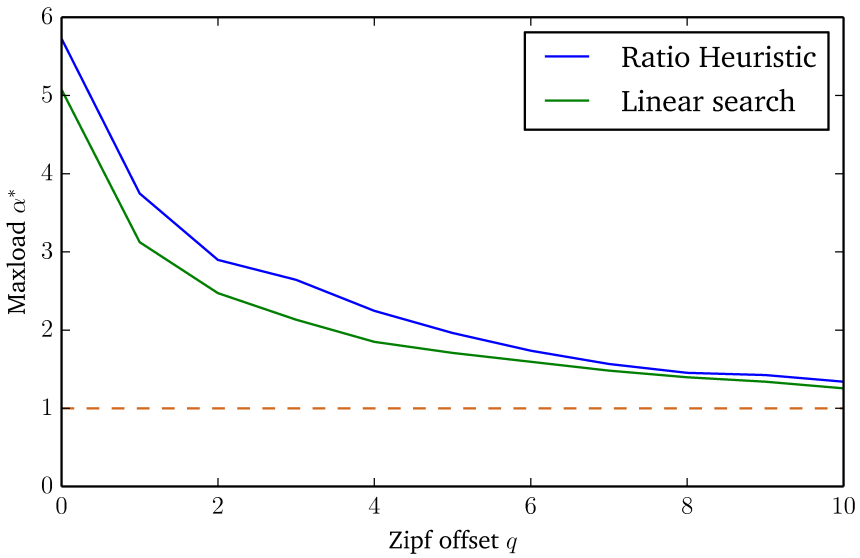

Figure 5.3: A comparison of the ratio heuristic and the linear search on the standard problem instance with a range of Zipf offsets, using three parts.

Figure 5.3 clearly shows that the linear search method consistently outperforms the ratio heuristic, so it is the method we will use in future. Furthermore, the ratio heuristic, while performing nearly as well as the linear search, tends to pick partitions with far larger initial parts, which gives a much larger product $|A_1||A_2|\ldots|A_t|$ and hence a longer running time when we actually compute the maximum load.

5.6 Evaluation: t parts

A comparison of the uniform random data layout using one, two and three parts on the standard problem instances is shown in Figure 5.4.

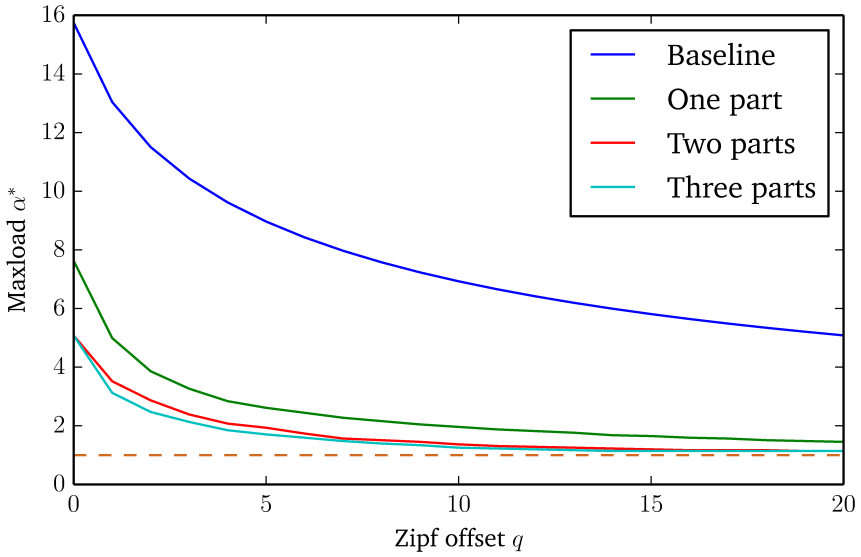

Figure 5.4: A comparison of the maximum load estimates found using $t = 1, 2, 3$ parts for the uniform random data layout, compared with the deterministic baseline, for the standard problem instance with a range of Zipf offsets q.

Here we can see that using three parts offers little additional improvement in the majority of cases as compared to using only two parts, so it appears there would be little benefit to consider any $t > 3$. Indeed, Figure 5.5 illustrates that increasing t offers diminishing returns, with most of the problem instances considered seeing no improvement for $t = 5$ over $t = 4$.

Additionally, for large t the time required to compute the maximum load becomes prohibitively long, so there is little reason to use more than three parts in most cases.

Figure 5.4 demonstrates the effectiveness of the uniform random layout. We can see that, compared with the deterministic baseline, the uniform random layout offers a very good solution with high probability, despite assuming nothing about the particular item demands.

We have seen in this chapter that a more refined analysis, which provided much tighter bounds on the worst-case maximum load from a uniform random data layout, is possible once we have partitioned the demand appropriately. We presented two methods for choosing such a partition: the ratio heuristic, which offered fair performance, and was able to decide on a partition very quickly; and, the linear search, which demonstrated superior performance at the expense of a longer running time.

It was also shown that by increasing the number of parts t used in the partition, both the level of accuracy of the estimate and the running time would increase. It was seen to be impractical to

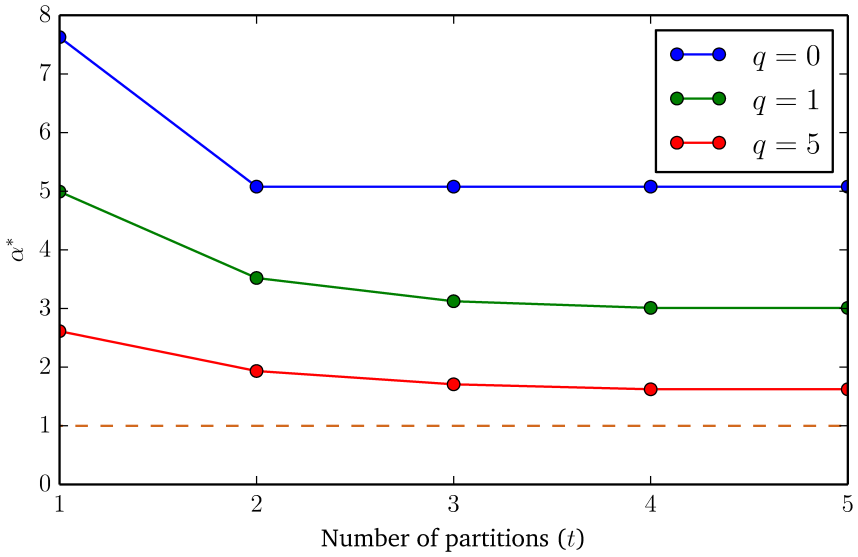

Figure 5.5: The diminishing returns from increasing the number of parts t used when computing the maximum load estimate for the standard problem instance, for a range of Zipf offsets q. Note how the maximum loads begin to "flatline" at around the $t = 3$ or $t = 4$ mark, whereby adding additional parts offers essentially no improvement to the maximum load.

run the algorithm for more than about $t = 3$ parts, and furthermore, diminishing returns were noticeable after this point (with most instances showing no further improvement from using 5 parts over the result obtained using 4 parts).

In the next chapter, we will see that the target goal assumed thus far—to attain a maximum load of $\alpha^* = 1$—is not achievable in general, and hence see just how good the performance of the uniform random data layout algorithm really is.

CHAPTER 6

Bound on performance

Thus far we have more or less assumed that the best performance is when maximum load $\alpha^* = 1$. That is, the most heavily laden disk has $\alpha^*\bar{L} = 1\bar{L} = \frac{\sum_{j=1}^{m} d_j}{n}$ incoming demand. However, for the demand distribution used thus far, this is generally unattainable (particularly for heavily skewed demand distributions).

Consider the standard problem instance, with Zipf offset $q = 0$. We find that $\bar{L} = \frac{1}{100} = 0.01$, and yet if we think about how the demand for the first item d_1 is routed, we see that $\bar{L}$ cannot be achieved. The demand d_1 is handled by at most $\ell = 3$ disks, hence the load on at least one of those disks has to be at least $\frac{d_1}{3}$. For our problem instance, $d_1 \approx 0.152$, so

$$\frac{d_1}{\ell} = \frac{0.152}{3} = 0.0507.$$

Given that $\bar{L} = \frac{\sum_{j=1}^{m} d_j}{n} = \frac{1.0}{100} = 0.01$, the load on these three disks must be at least $0.0507 = 5.07\bar{L}$, and hence no uniform data layout for this problem instance can ever do better than $\alpha^* = 5.07$.

We saw that using two parts in the analysis of the uniform random data layout gave us a maximum load estimate of 5.07, which is why increasing the number of parts t did not improve the result.

In this chapter, we aim to find better bounds on performance than just $\alpha^* \geq 1$, so that we can more accurately judge the performance of the uniform random data layout algorithm.

6.1 Approximate lower bound

We just saw that by looking at only the demand of the first, most popular, item for the standard problem instance with Zipf offset $q = 0$, we were able to obtain a tight bound on the performance of a data layout. By distributing the demand d_1 evenly across ℓ disks, we can find a tighter lower bound for α^* than the value of 1 assumed until now.

It could be possible for particular demand distributions that d_1 can be spread over ℓ disks without exceeding $\bar{L}$ demand per disk. Recall that we have an absolute lower bound of $\bar{L}$ units of demand per disk, which occurs when all units of client demand are shipped to the set of storage disks. Hence we define a lower bound $\widetilde{\alpha}$ as follows:

$$\widetilde{\alpha} = \max\left\{\frac{d_1}{\ell\bar{L}}, 1\right\}.$$

Plots now follow showing how the maximum load estimates α^* using the uniform random data layout on the standard problem instance compare to this lower bound $\widetilde{\alpha}$, which was calculated using the demand of only the first item, d_1.

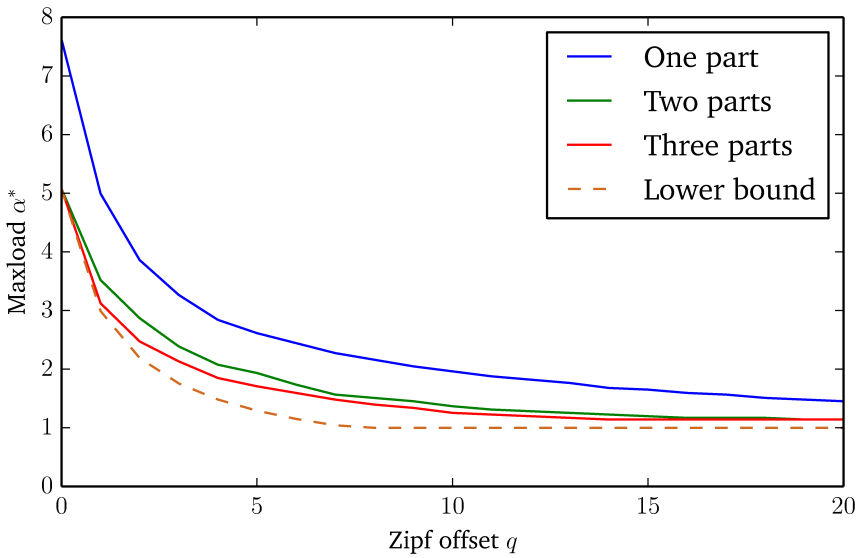

Figure 6.1: A comparison of the uniform random data layout using up to three parts, with the $\widetilde{\alpha}$ lower bound on performance. Run on the standard problem instance with a range of Zipf offsets q.

Figure 6.1 plots the maximum load estimates (as shown earlier; see Figure 5.4) against the lower bound $\widetilde{\alpha}$. As the number of parts t used in the analysis increases, the curves gradually tend towards this lower bound. The difference between the estimates and $\widetilde{\alpha}$ is shown in Figure 6.2.

As q increases, the Zipf distribution flattens out, which might explain why the maximum load estimates α^* deviate from the bound $\widetilde{\alpha}$. $\widetilde{\alpha}$ is calculated using only the first item's demand, where we assume that the maximum load is attained by one of the disks to which u_1 is assigned. That is, we assume that the set $S \subseteq U$ which attains the maximum load in $\max_{S \subseteq U} \frac{d(S)}{L(S)}$ is the set $S = \{u_1\}$. However, for less-skewed distributions, it might be possible that S could include other items, meaning $\widetilde{\alpha}$ is a lower bound on true best possible maximum load that could occur for our problem instance.

In the following section we aim to improve this approximation to provide a tight lower bound on performance.

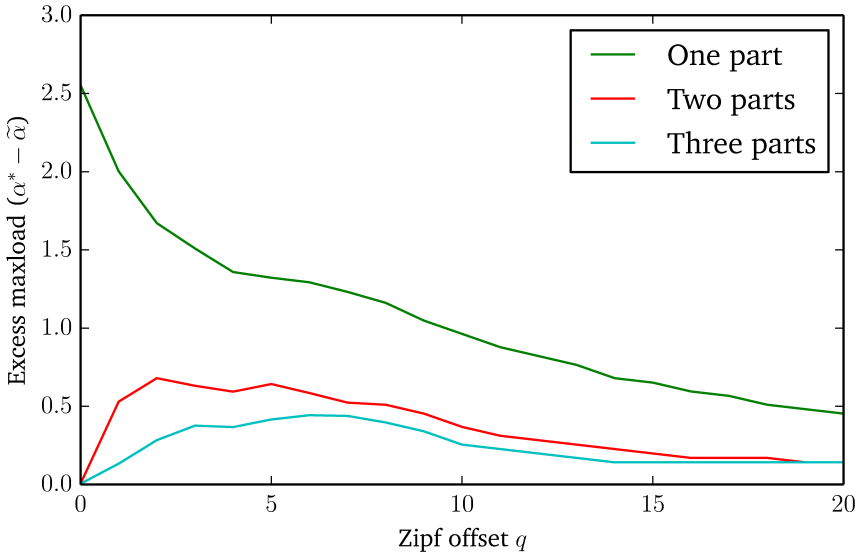

Figure 6.2: A tighter performance analysis for the uniform random data layout using up to three parts on the standard problem instances. The lines plotted show the lower bound $\widetilde{\alpha}$ subtracted from the maximum load estimate α^*, i.e. how far from optimal the random data layouts are. A difference of zero means the layout is perfect.

6.2 Mixed integer program formulation

Rather than simply guess which subset of the data items $S \subset U$ attains the maximum load, we will now actually solve each of the standard problem instances outright, and find what the true optimum performance is. That is, given knowledge of the exact item demands ahead of time, what is the best data layout we can devise such that the maximum load across the disks is minimised?

For this, we design a mixed integer program (see MIP 1 on page 43) that will minimise the maximum load, subject to the constraints of uniform data layouts that were defined in section 3.1. To solve this program, we used the Gurobi Optimizer mixed integer programming solver [Opt13].

We have our usual sets of items U and disks V, with a complete set of edges between them $E = U \times V$. The parameters ℓ and k are given, which fix the number of copies made of each item and the storage capacity of the disks, respectively. The demand d_u for each item $u \in U$ is assumed to be fixed and known.

Finally, we introduce the variables in the mixed integer program. For each edge $(u, v) \in E$, we define two variables x_{uv} and y_{uv}. x_{uv} is a binary variable which indicates whether item u is to be copied onto disk v. We introduce constraints (6.2) and (6.3), which restrict assigning each item to at most ℓ disks and each disk to at most k items. The variable y_{uv} represents the demand that is sent across the edge (u, v). Constraint (6.4) forces each item u to ship all of its demand d_u to the disks. Constraint (6.5) limits the demand on each edge. If $x_{uv} = 0$

Sets

U		set of data items
V		set of storage disks
E		set of edges ($E = U \times V$)

Parameters

ℓ		number of copies made of each item
k		storage capacity of each disk
d_u	$\forall u \in U$	demand for item u

Variables

x_{uv}	$\in \{0,1\}$	$\forall (u,v) \in E$	indicator for whether u is to be copied onto v
y_{uv}	≥ 0	$\forall (u,v) \in E$	demand for u that is sent to v
x_{uv}	≥ 0		maximum load

$$\text{minimise} \quad z \tag{6.1}$$

$$\text{subject to} \quad \sum_{v \in V} x_{uv} \leq \ell \qquad \forall u \in U \tag{6.2}$$

$$\sum_{u \in U} x_{uv} \leq k \qquad \forall v \in V \tag{6.3}$$

$$\sum_{v \in V} y_{uv} = d_u \qquad \forall u \in U \tag{6.4}$$

$$y_{uv} \leq x_{uv} d_u \qquad \forall (u,v) \in E \tag{6.5}$$

$$\sum_{u \in U} y_{uv} \leq z \qquad \forall v \in V \tag{6.6}$$

$$x_{uv} \in \{0,1\} \qquad \forall (u,v) \in E \tag{6.7}$$

$$y_{uv} \geq 0 \qquad \forall (u,v) \in E \tag{6.8}$$

$$z \geq 0 \tag{6.9}$$

Mixed Integer Program 1: Program that finds the optimum uniform random layout for the given problem instance, and returns its maximum load.

then u has not been assigned to v and hence we cannot send any demand across that edge (so $0 \leq y_{uv} \leq x_{uv}d_u = 0 \implies y_{uv} = 0$). Otherwise, $x_{uv} = 1$ so we can ship at most d_u units of demand across the edge (u, v) (so $0 \leq y_{uv} \leq x_{uv}d_u = d_u$). Lastly we have the variable z representing the maximum load across the disks. The objective function minimises z, so we constrain z to be no smaller than the load on each disk (constraint (6.6)), so now effectively $z = \max_{v \in V} \sum_{u \in U} y_{uv}$.

This program will find the best possible maximum load for the problem instance, hopefully giving an even tighter lower bound on performance.

Running it for the standard problem instances proved that the bound given by $\widetilde{\alpha}$ earlier was tight (at least for those particular instances); the bound found by MIP 1 in every case resulted from either spreading the first item's demand over ℓ disks, or from spreading the entire demand over all disks (giving a maximum load of 1). Hence we can effectively regard the bound $\widetilde{\alpha}$ as tight.

6.3 Evaluation

As we just saw in the previous section, Figure 6.2 gives a true indication of the extent to which the random data layout is suboptimal. We can see that using three parts in the analysis means we never exceed the optimal maximum load by more than 0.5.

In all earlier chapters of this work, we have considered the maximum load in terms of a multiple of the average load $\bar{L}$, with the assumption that $\bar{L}$ is the goal for which we should aim. We now know that the optimum maxload is given by $\widetilde{\alpha}\bar{L}$, and so it is with respect to this value that we wish to evaluate the uniform random data layout.

In order to learn something about the kind of approximation to the true optimum we get, we will plot the ratio for the standard problem instance:

$$\Delta := \frac{\text{maxload estimate}}{\text{lower bound}} = \frac{\alpha^*\bar{L}}{\widetilde{\alpha}\bar{L}} \frac{\alpha^*}{\widetilde{\alpha}}.$$

This will suggest something about the nature of the performance of the uniform random data layout, and we can posit that it is a Δ-approximation.

Figure 6.3 illustrates that the uniform random data layout using three parts is able to give a 1.4-approximation for the standard problem instances. This is a very good result, and the range of Zipf offsets used (simulating demand distributions with various levels of skewness) suggest that this approximation factor of 1.4 is likely very close to the true value. Unfortunately, we do not currently have a way of generalising this approach to all problem instances.

It is worth reiterating at this point that the maximum load estimate we have computed is in fact an upper bound on the worst-case maximum load, for all possible assignments of demands to items. Hence this approximation ratio Δ is a way of bounding the worst-case maximum load performance, *for all demand assignments*. This is a very strong statement, and implies that, for

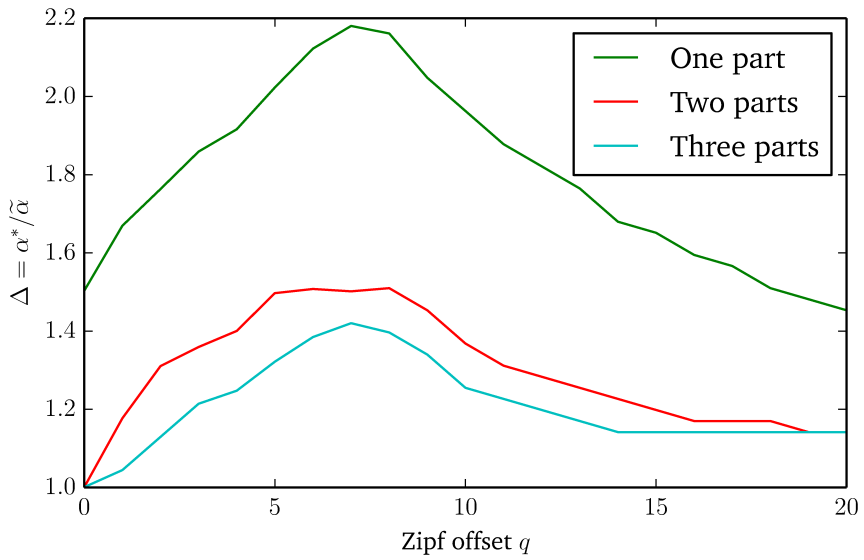

Figure 6.3: Approximation ratios $\Delta = \frac{\alpha^*}{\widetilde{\alpha}}$ for a the standard problem instance with a range of Zipf offsets q.

these problem instances, and with very high probability, any assignment of demands to items will give a maximum load of no more than 1.4-times the optimum—an excellent result.

CHAPTER 7

Maximum coverage

We now leave behind all thoughts of maximum load and the load balancing objective, and consider instead the maximum coverage objective defined in section 3.1. Recall this was defined to be the proportion of client demand that is satisfiable by a given data layout, where we place a hard restriction on the load on the storage disks of $\bar{L}$ (i.e. the total demand averaged over all disks).

Here we will assume that each disk can accept no more than $\bar{L} = \frac{\sum_{j=1}^{m} d_j}{n}$ demand (i.e. the smallest load such that it should still be possible to assign all demand). Assuming we still require exactly ℓ copies made of every data item, and use storage disks with capacity k, we now wish to investigate how well the uniform random data layout performs on unseen demand distributions.

The *coverage* λ is defined as

$$\lambda = \frac{\text{demand delivered}}{\text{total demand}}.$$

Previously, we saw how to measure the maximum load by constructing a network flow graph (see section 3.2, particularly Figure 3.3). The maximum coverage can be computed in a similar way. See Figure 7.1. The only difference we make to the network flow instance is that the edges connected to the sink node t are given capacity $\bar{L}$ (signifying the strict load limit). The maximum flow f in this graph gives the maximum deliverable demand. Hence the coverage becomes

$$\lambda = \frac{|f|}{\sum_{j=1}^{m} d_j}.$$

7.1 Coverage derived from the maximum load

In the previous chapters, we build a framework for analysing the maximum load that we can expect from the uniform random data layout algorithm. The resulting method developed is able to give an upper bound (with high probability) on the worst-case maximum load for the

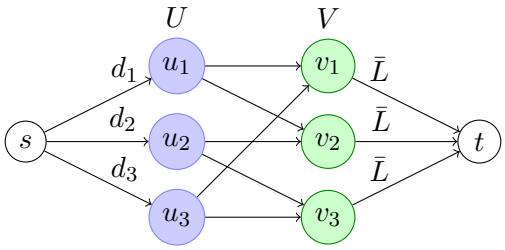

Figure 7.1: An example network flow instance for the maximum coverage problem.

given demand distribution. By worst-case, we mean the maximum load resulting from the worst possible assignment of the demands to the data items (and hence, all such assignments).

In this section we will reuse this bound to obtain a similar statement about the worst-case maximum coverage. It is worth clarifying at this point exactly what this bound tells us. Having fixed the problem instance and demand distribution for the analysis, we can compute the maximum load bound α^* exactly as we did before. This is then used to give a corresponding bound on the maximum coverage, λ. The guarantee implied by this bound is that the uniform data layout chosen that is chosen randomly will, with high probability, satisfy the following property: for any assignment of the demands to items, there is always a way of routing the client demand to storage disks such that we can cover at least a λ-fraction of the total demand, without any disk ever having a load of more than $\bar{L}$.

In the rest of this chapter we show exactly how this bound on the maximum coverage can be derived from the equivalent bound on the maximum load.

Initial lower bound

Up until now, we have allowed the disks to accept unbounded demand, with the aim of minimising the maximum load while still catering to the entire client demand. If the load on the most heavily-laden disk is given by $\alpha^*\bar{L}$ where $\alpha^* > 1$, then we can find a (loose) lower bound on the coverage of that layout.

A maximum load of α^* tells us that every storage disk accepts at most $\alpha^*\bar{L}$ demand. Limiting the load on each disk to $\bar{L}$ means we can only serve a $\frac{1}{\alpha^*}$-fraction of the demand that was assigned to each disk. This is clearly a gross underestimation of the actual coverage; we saw in chapter 6 that the disks that attain the maximum load are the ones to which the most popular item is assigned (i.e. in many cases, only ℓ of the disks have a load greater than $\bar{L}$, so assuming that every disk has load $\alpha^*\bar{L}$ is far too pessimistic). This is shown in Figure 7.2.

Item exclusion

Given that the most popular item is frequently the cause of the high maximum load estimate (and hence low coverage estimate), it might be possible to improve the coverage estimate by ignoring the first item entirely. If we just remove the demand d_1 from our analysis entirely, and

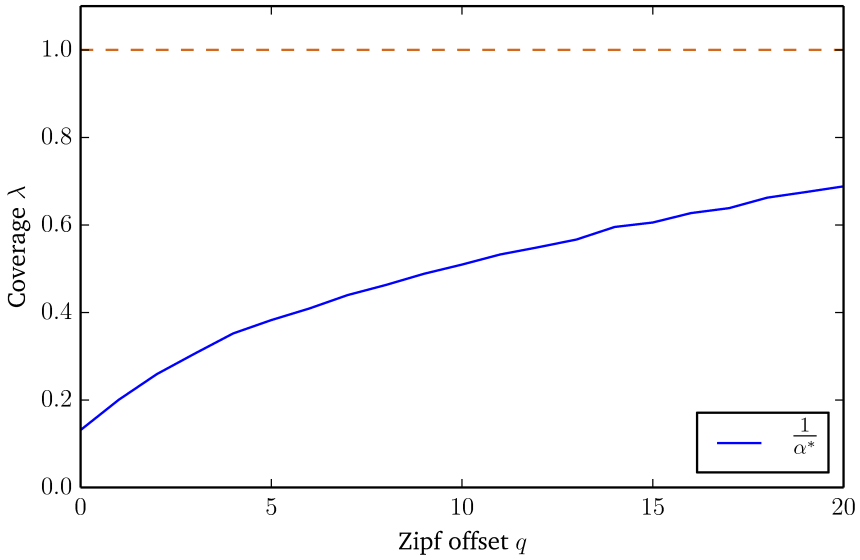

Figure 7.2: An initial lower bound for coverage, assuming that the coverage $\lambda = \frac{1}{\alpha^*}$, where the maximum load estimate α^* is from the random uniform data layout analysis with one part on the standard problem instance. The best possible coverage is $\lambda = 1$, indicated by a dashed orange line.

estimate the maximum load on our uniform random data layout by assuming d_1 does not exist, it could be possible that the lower maximum load α^* found improves the coverage estimates for the rest of the disks.

We can extend this idea to the first a data items. If we exclude the first a items from the analysis (e.g. by reassigning the first a items to have 0 demand: $d_1, \ldots, d_a \leftarrow 0$), we obtain a maximum load estimate $\alpha^*(a)$. This represents a multiple of the average load $\bar{L}$, with respect to the modified demand distribution. Therefore, we can serve a $\frac{1}{\alpha^*(a)}$-fraction of total modified demand. That is, $\frac{1}{\alpha^*(a)} \sum_{j=a+1}^{m} d_j$. As a proportion of the total demand, this gives us the coverage λ:

$$\lambda = \frac{\frac{1}{\alpha^*(a)} \sum_{j=a+1}^{m} d_j}{\sum_{j=1}^{m} d_j} = \frac{1}{\alpha^*(a)} \left(1 - \frac{\sum_{j=1}^{a} d_j}{\sum_{j=1}^{m} d_j}\right),$$

i.e. the coverage is $\frac{1}{\alpha^*(a)}$-times the proportion of included demand.

Figure 7.3 shows how the coverage λ varies as we change the number of item a that are excluded. The maximum coverage in each case is marked with an "$\times$".

By comparing the maximum coverage with that at $a = 0$ in each case, we can see the vast improvement over our initial estimation. If we consider the curve corresponding to Zipf offset $q = 0$, the coverage improves from 0.131 to 0.311 by completely ignoring the first 20 items! This demonstrates not only how bad our first estimate was, but suggests we can improve on this result even further. The $q = 0$ demand distribution was very heavily skewed, with a large portion of the distribution's mass included in the first few items. In fact, the first $a = 20$ items

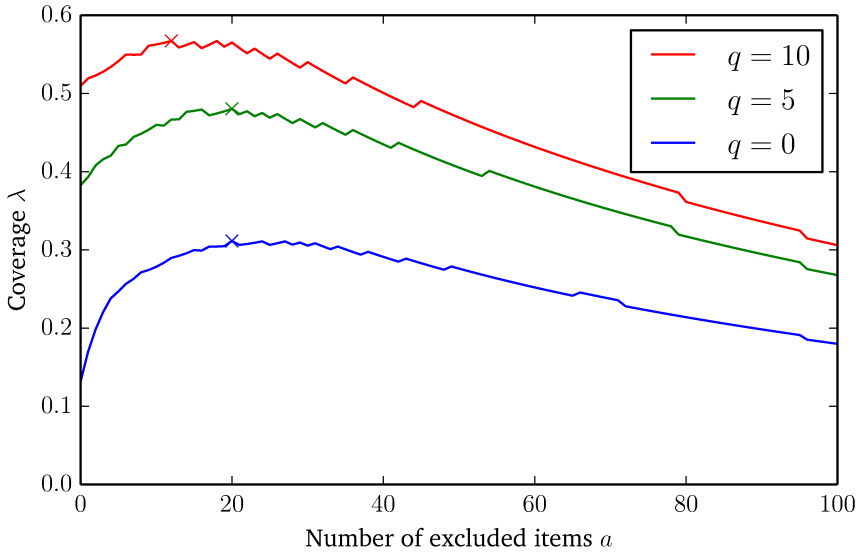

Figure 7.3: The coverage obtained by excluding the first a data items. Run on the standard problem instances with Zipf offsets $q = 0, 5, 10$. Only the values of $a \leq 100$ have been plotted. The maximum loads were estimated using one part.

represents 55.5% of the total demand for the $q = 0$ problem instance. By excluding the first 20 items, we have excluded more than half of the demand, so we can only hope to achieve a coverage of no more than $\lambda = 0.445$ with this technique.

This motivates the analysis we present next, where we aim to recover some of this excluded demand.

Demand truncation

Rather than completely discarding the demand of the first a data items, we now consider instead *truncating* that demand. The initial motivation for excluding the high-demand items was from our previous observations that the item with the highest demand is often solely responsible for causing a high maximum load (and hence low coverage). By excluding the first item, we hope that the second item does not exhibit this same trait, giving us a lower maximum load. However, if d_2 is low enough that it can be spread across ℓ disks and not increase the maximum load overly much, there is no reason why we cannot deliver d_2 units of the first item's demand as well, for little or no added penalty.

This concept extends in a straightforward way to an arbitrary number of items a. Instead of setting the first a items' demands to zero ($d_1, \ldots, d_a \leftarrow 0$) as we did before, we will instead truncate them to the next highest demand, d_{a+1} (so $d_1, \ldots, d_a \leftarrow d_{a+1}$).

Figure 7.4 gives a visualisation of the demand that is recovered from truncating the demands rather than ignoring those items completely. The figure depicts the Zipf distribution with parameters $s = 1$ and $q = 0$, for the case where $a = 5$. The red region shows the demand that is

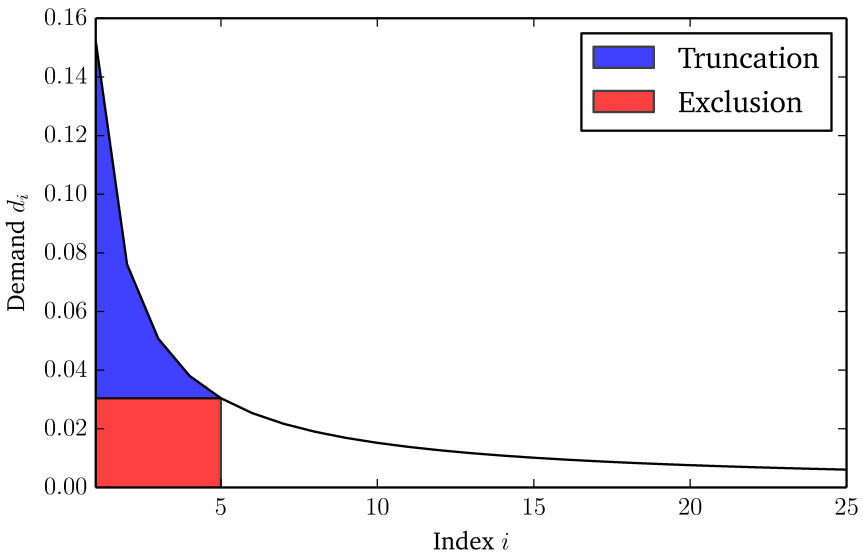

Figure 7.4: The demand recovered by truncation, as compared to item exclusion. The plotted curve shows the demands of the first 25 items when the demand follows a Zipf distribution with $s = 1$ and $q = 0$. If we set $a = 5$, then the blue shaded region illustrates the demand lost through truncation, while the red area shows the *additional* demand lost through item exclusion.

discarded during item exclusion, which we can recover using the truncation method. The blue region is the demand that is sacrificed by both methods. We can see in Figure 7.5 a comparison of the total proportion of the demand that is lost using each method, plotted against the number of ignored items a. It shows that while truncation is able to reclaim about 10–15% of the total demand, a very large portion of the demand is still lost, even for very small values of a.

The improvement to the maximum coverage estimate by using demand truncation is shown in Figure 7.6. Each curve plots how the coverage estimate varies with the number of items a that we ignore, for different problem instances. The dotted lines show the old results (from Figure 7.3) that are obtained by excluding the first a items. The solid lines show the new estimates using demand truncation. We can see that in each problem instance tested, we obtain a significant improvement to the maximum coverage estimate. For instance, when $q = 0$ we find that we can truncate the first 43 items to obtain a coverage of 0.394, which allows us to cover about 8% more demand than what the previous result of 0.311 would suggest.

This gives us a simple technique to get a good lower bound on the maximum coverage offered by the uniform random data layout. We can perform a linear search along the curve, as plotted in Figure 7.6, stopping once a sensible termination criterion is met (e.g. explore a fixed number of points beyond the maximum found so far, or explore until the difference between the current value and the maximum found becomes too large). For all results presented from here onwards, we terminate after exploring the next 10 points and not finding that we have updated the maximum.

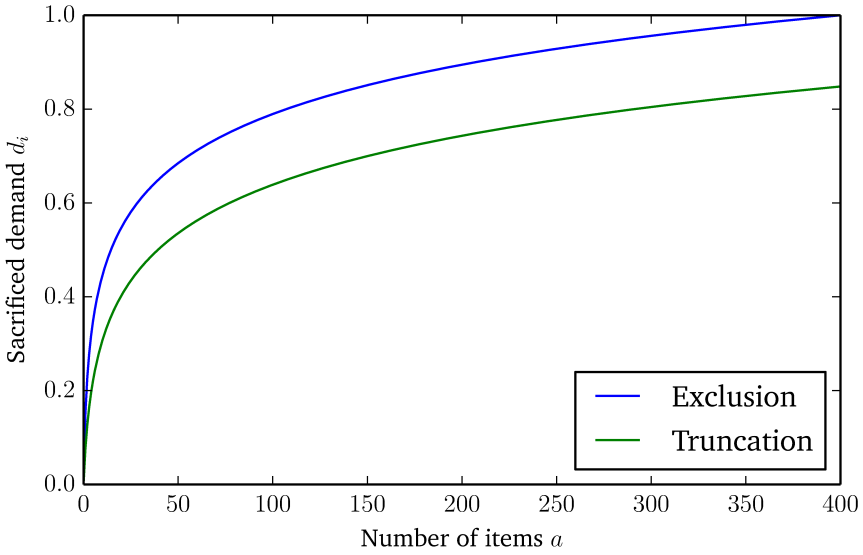

Figure 7.5: The total proportion of the demand that is sacrificed by the exclusion and truncation methods, against the number of ignored items a. The demands follow a Zipf distribution with $s = 1$ and $q = 0$.

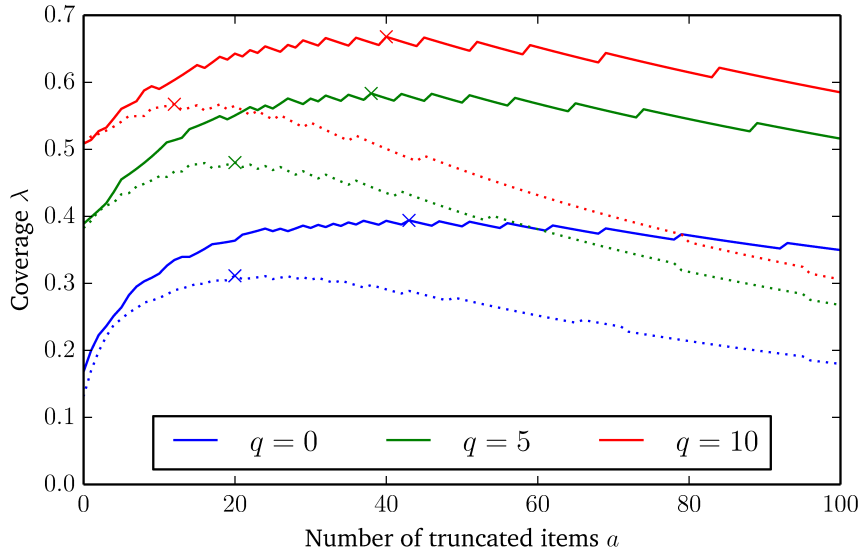

Figure 7.6: The solid lines plot the coverage obtained by truncating the first a item demands to the value of d_{a+1}. The dotted lines are to show the improvement over item exclusion, where each dotted line plots the item exclusion result for the instance of the matching colour. Run on the standard problem instances with Zipf offsets $q = 0, 5, 10$. The maximum loads were estimated using one part.

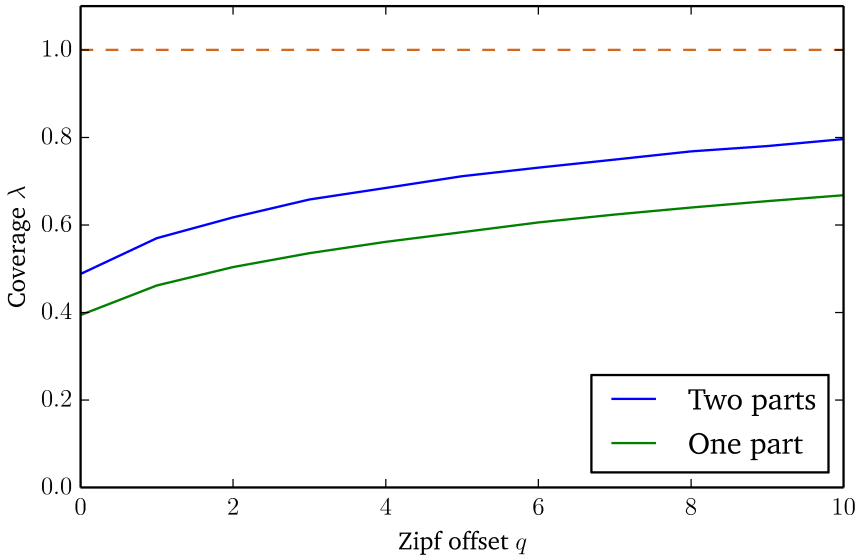

Figure 7.7: Maximum coverage estimate for the uniform random data layout using a maximum load analysis with one and two parts. Run on the standard problem instances for a range of Zipf offsets q.

The results of the linear search method are shown in Figure 7.7, with separate curves for using one and two parts in the maximum load computation. An analysis using three or more parts is not feasible due to the large number of times we need to run the algorithm—the maximum coverage generally occurs at about $a = 30$, meaning we need to re-run the maximum load linear search algorithm approximately 30 times to compute each data point—which takes far too long when using more than two parts.

7.2 Upper bound

In chapter 6, we solved the standard problem instances optimally using a mixed integer program (and an approximation scheme derived from the highest value in the demand distribution) in order to bound the performance we could hope to expect from our uniform random problem instance. In this section, we will design a similar procedure to obtain an upper bound on the maximum coverage objective.

Just as we were unable to design a data layout to lower the maximum load on the storage disks any more than the optimum found in chapter 6, so too can we not cover more than some optimum portion of the client demand for our standard problem instances. That is, it is not always possible to achieve a demand coverage of $\lambda = 1$.

To find the optimum coverage for a given problem instance, we will lose the assumption of unknown demands and instead assume the distribution $d_1, \ldots, d_m$ is fixed. We can then write a mixed integer program to solve the instance exactly.

Sets

U		set of data items
V		set of storage disks
E		set of edges ($E = U \times V$)

Parameters

ℓ		number of copies made of each item
k		storage capacity of each disk
d_u	$\forall u \in U$	demand for item u

Variables

x_{uv}	$\in \{0, 1\}$	$\forall (u, v) \in E$	indicator for whether u is to be copied onto v
y_{uv}	≥ 0	$\forall (u, v) \in E$	demand for u that is sent to v
x_{uv}	≥ 0		maximum load

$$\text{maximise} \quad \sum_{(u,v) \in E} y_{uv} \tag{7.1}$$

$$\text{subject to} \quad \sum_{v \in V} x_{uv} \leq \ell \qquad \forall u \in U \tag{7.2}$$

$$\sum_{u \in U} x_{uv} \leq k \qquad \forall v \in V \tag{7.3}$$

$$\sum_{v \in V} y_{uv} \leq d_u \qquad \forall u \in U \tag{7.4}$$

$$\sum_{u \in U} y_{uv} \leq \bar{L} \qquad \forall v \in V \tag{7.5}$$

$$y_{uv} \leq x_{uv} d_u \quad \forall (u, v) \in E \tag{7.6}$$

$$x_{uv} \in \{0, 1\} \quad \forall (u, v) \in E \tag{7.7}$$

$$y_{uv} \geq 0 \qquad \forall (u, v) \in E \tag{7.8}$$

Mixed Integer Program 2: Program that finds the optimum uniform random layout for the given problem instance, and returns its maximum coverage.

MIP 2 on page 53 gives the true optimum coverage (c.f. MIP 1, page 43). We again define variables $x_{uv} \in \{0, 1\}$, a matching indicator, and $y_{uv} \geq 0$, representing the flow of demand, for all data items $u \in U$ and storage disks $v \in V$. We no longer need the variable z (or any of its related constraints). We make the additional following changes to the constraints from MIP 1. The demand sent from each data item $u \in U$ (given by $\sum_{v \in V} y_{uv}$; see constraint (7.4)), no longer strictly equals d_u; we wish to allow the possibility of only satisfying part of the demand. Similarly, we introduce constraint (7.5) to limit the incoming demand $\sum_{u \in U} y_{uv}$ on each disk $v \in V$ to at most $\bar{L}$. Finally, our objective function changes to maximise the total demand covered, $\sum_{(u,v) \in E} y_{uv}$, see (7.1).

MIP 2 gives us an upper bound on the performance we can expect from our uniform random data layout on the maximum coverage objective.

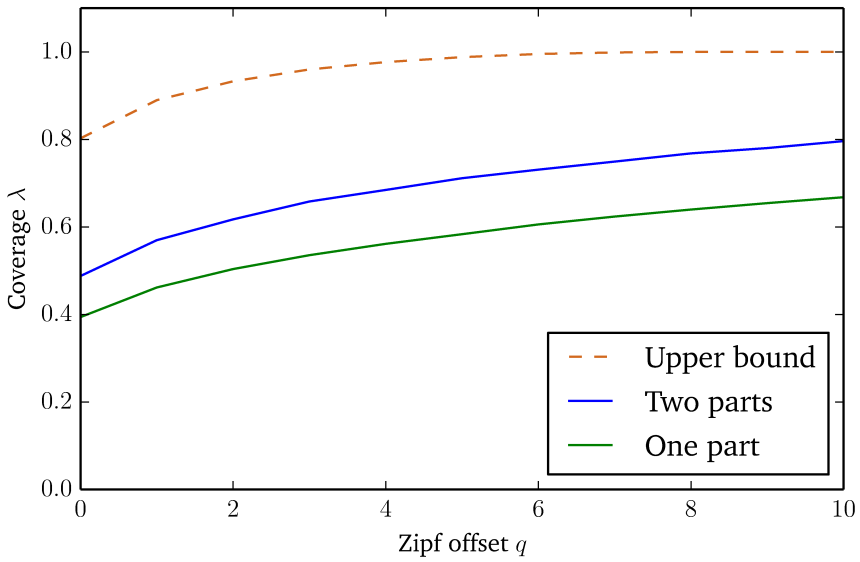

Figure 7.8: Maximum coverage estimates for the uniform random data layout using a maximum load analysis with one and two parts, compared with the upper bound provided by MIP 2. Run on the standard problem instances for a range of Zipf offsets q.

Figure 7.8 plots the results found earlier in section 7.1 against the results obtained from solving MIP 2. We can see that, as compared to Figure 6.1 on page 41, the performance of the uniform random data layout is nowhere near optimal. The approximation ratio Δ for the maximum coverage problem on these problem instances is shown in Figure 7.9, where Δ is the estimated coverage divided by the optimum coverage found using MIP 2. We can see that the ratio obtained is very poor, with a guarantee from the two-part analysis of only 60% of the optimum coverage.

This is almost surely due to an inaccurate estimate from our crude analysis of the maximum coverage, rather than representative of poor performance of the layout itself. In our analysis we completely disregarded huge swaths of client demand, assuming that it was not possible for

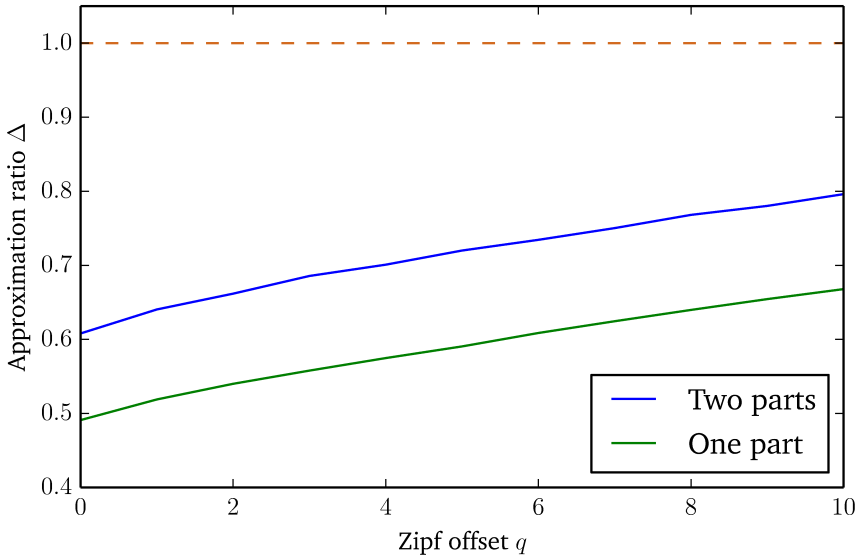

Figure 7.9: Maximum coverage approximation ratios for the uniform random data layout using a maximum load analysis with one and two parts, compared with the upper bound provided by MIP 2. Run on the standard problem instances for a range of Zipf offsets q. The ratio Δ plotted is the ratio of the maximum coverage estimation over the optimum maximum coverage.

them to be assigned. Although this was mitigated slightly by truncating rather then completely excluding the item demands, this analysis is in no way tight.

7.3 Simulations

As explained in the previous section, there is strong reason to believe that the performance on the maximum coverage objective is in fact far better than that suggested by the bounds we have obtained. In order to provide some evidence of this, in this section we will run a series of simulated experiments as "proof" that this bound is not tight.

Recall the meaning behind the curves shown in Figure 7.8 for one and two parts. The number of parts indicates the parts in the partition used for the maximum load analysis, but more importantly recall the guarantee that each curve implies. Given a fixed demand distribution (the graph shows standard problem instances for various values of the Zipf offset q), with very high probability, the maximum load resulting from any possible assignment of those demands to the data items is bounded by α^*. This translates to a similar statement for the maximum coverage: with very high probability, for every possible assignment of demands we can achieve a coverage of at least λ.

We have seen that by increasing the number of parts t used in the analysis we can tighten this bound considerably. We can perform some experiments that suggest that this bound can be tightened even further.

For each problem instance we wish to consider, we build the demand distribution, and then run a number of trials. In each trial we generate a uniform random data layout as per our algorithm. There are far too many different assignments to demands, so checking every permutation is not feasible. Instead, we will spend some amount of time trying to find the worst demand assignment such that the maximum coverage is a minimum. Call this the number of *repeats* for each trial (we used 100 repeats); the minimum value is the value recorded for this trial. We then generate a new random uniform data layout and repeat (we used 50 trials). The maximum coverage in each case was computed using the linear program LP 3. This is plotted in Figure 7.10 as a point with error bars. Each plotted point gives the average for all 50 trials, while the error bars denote the maximum and minimum values found. Hence the error bars illustrate the spread of the values, and can be treated as an approximation of the probability distribution.

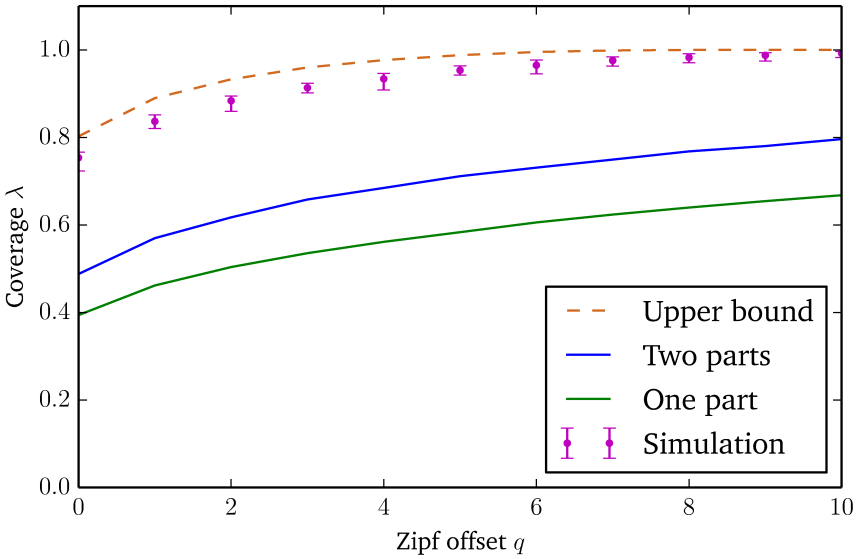

Figure 7.10: Maximum coverage approximation ratios for the uniform random data layout using a maximum load analysis with one and two parts, compared with the upper bound provided by MIP 2. Run on the standard problem instances for a range of Zipf offsets q. The ratio Δ plotted is the ratio of the maximum coverage estimation over the optimum maximum coverage.

It is worth reiterating that the plotted values represent the coverage that is obtained when the worst possible assignment of demands to items is used (where by "worst" we mean the worst that we could find in 100 repeats). Hence the minimum error bar gives the worst coverage that we could find for that problem instance (from the 50 different random data layouts we tried). That the plotted points are so close to the optimum dashed line suggests that the data layout algorithm in general is able to achieve excellent performance, regardless of what our lower bounds might suggest.

The error bars give the range of the worst-case performance that we could find for the 50 random layouts generated. The fact that the error bars are so small indicates that the worst-

Sets

U		set of data items
V		set of storage disks
E		set of edges in data layout

Parameters

d_u	$\forall u \in U$	demand for item u
$\bar{L}$		the average load ($\bar{L} = d(U)/\|V\|$)

Variables

y_{uv}	≥ 0	$\forall (u,v) \in E$	demand for u that is sent to v

$$
\begin{aligned}
\text{maximise} \quad & \sum_{(u,v)\in E} y_{uv} \\
\text{subject to} \quad & \sum_{v\in\delta(u)} y_{uv} \leq d_u \quad \forall u \in U \\
& \sum_{u\in\delta(v)} y_{uv} \leq \bar{L} \quad \forall v \in V \\
& y_{uv} \geq 0 \quad \forall (u,v) \in E
\end{aligned}
$$

Linear Program 3: Program that finds the maximum coverage for the given data layout.

case performance is very tightly concentrated. Also note how close these are to the dashed curve that represents the upper bound. This bounds the best-case performance for each (that is, the demand assignment that allows for the best possible coverage to be attained), which shows that the variation in maximum coverage by choosing different demand assignments is actually very small.

While this simulation is by no means a proof, it does strongly suggest that the bounds found were not tight. At no point were we ever able to find a data layout with an assignment of demands that produced a maximum coverage that was even close to the bounds. In fact, the values found were only slightly worse that the optimum coverage.

The conclusions that we can draw from this are twofold. Firstly, we have convincing evidence that the uniform random data layout algorithm gives very robust data layouts that give very good demand coverages for almost any assignment of demands. Secondly, this same evidence suggests that the bounds on the maximum coverage derived from our earlier research into the load balancing objective are not at all tight, and so further study is still needed.

CHAPTER 8

Discussion

In this chapter, we discuss the main theoretical and experimental results presented in this thesis, analysing the implications and importance of these results. This is followed by a summary of the open problems and further research prompted by this work.

8.1 Results

Load balancing objective

In chapter 3 we outlined the uniform data layout problem with unknown demands, and presented the uniform random data layout as a potential solution to it. We also detailed an initial framework for analysing the layout with regard to the load balancing objective, for any fixed demand distribution that is given. The probabilistic approach of the analysis inspires an obvious algorithm for estimating the maximum load for the uniform random data layout on any given problem instance. This analysis gives the first known framework for estimating the maximum load of this data layout, without resorting to repeated simulation experiments as a way of identifying the worst-case maximum load.

We saw that the upper bound on the maximum load found from this analysis was demand-assignment agnostic. This allows us to bound the maximum load for any assignment of demand to data items, with high probability. From this we can show that the performance of the uniform random data layout algorithm is exceptionally good, no matter which particular items happen to have the highest demand.

The analysis was extended in chapter 5 by partitioning the client demand into arbitrarily many parts, which was seen to offer considerable improvement in the maximum load estimates; however, the time complexity of the estimation was seen to be $\mathcal{O}(m^{t+1})$ for a number of parts t, which—while not overwhelmingly bad—still means running this algorithm for anything other than very small t becomes impractical. Additionally, we presented two different methods—initially designed for a single part, but later extended to an arbitrary number of parts t—for

choosing a good demand partition to use: the ratio heuristic, and the linear search. The latter approach was seen to yield superior results, but at the cost of a larger running time.

The improved results from having a more refined analysis allowed a more accurate evaluation of the uniform random data layout. In chapter 4 we discussed the Zipf family of demand distributions as offering a suitably diverse range to illustrate the performance of data layout algorithms, with enough confidence in the results to generalise them more broadly. We saw that the layout offered at least a twofold improvement over a naïve deterministic layout that served as our baseline, which was a good start. Considering variations on the standard problem instances used (e.g. by adjusting the number of copies ℓ made of each data item, and by increasing the problem size) allowed us to conclude not only that we can expect the effectiveness of the uniform random data layout to persist across different problem instances, but also that the layout benefits greatly from the increased data redundancy that comes with increasing ℓ.

These results were improved upon further in chapter 5 where we partitioned the demand to obtain a better estimate. We found that using three parts showed at least a threefold improvement over the baseline. In chapter 6 we were able to tighten the bounds on the best possible performance on the load balancing objective that can be attained by any uniform data layout (even with prior knowledge of the demand distribution). The mixed integer program MIP 1 was used to solve a range of problem instances exactly, where we saw that the three-part analysis of the uniform random data layout was able to achieve an approximation factor of 1.4 for the problem instances tested. This is an incredible achievement, especially considering that the random layout is completely oblivious to the item demands or any expectation of their demands, and just assigns them to disks randomly. This implies that if one was given perfect prior knowledge of the demand distribution and the same set of requirements about the number of copies made per item and the capacity limit of each disk, one could expect to lower the maximum load across the storage disks by about only 30%.

This evaluation clearly demonstrates the efficacy of the uniform random data layout for the load balancing objective. In a domain where there is no prior knowledge available of the demand distribution of the items, assigning them to disks uniformly at random is a very effective initial strategy to employ, at least until more knowledge about the demands is available.

Maximum coverage objective

The other problem we considered was that of maximising the demand coverage. This was explored in chapter 7 where we derived a coverage estimate from the maximum load as estimated through the methods outlined in the earlier chapters. We saw that the most straightforward estimate given by $\frac{1}{\alpha^*}$ was very pessimistic, which led to trying two new approaches: excluding the highest-demand items, and truncating the highest demands. The latter was seen to give far better estimates; however, compared to the theoretical maximum coverage, as found by solving each instance with the mixed integer program MIP 2, the results were not very impressive. We suspect more so that a tighter analysis is required than that the expected coverage of the

uniform random data layout is actually that low. Experiments were conducted by simulation random data layouts and possible demand assignments, which offered some evidence supporting this. Further research into this is required to improve the lower bound on the maximum load that we give.

8.2 Open problems

This section outlines the opportunities for future work and areas in need of further study which became apparent during this research.

Proportional random data layouts

This problem was briefly mentioned in chapter 1, but represents the next logical step in this body of work. We began with the assumption that the demand distribution is entirely unknown. A far more realistic assumption is that *something* is known about the item demands. Consider our original motivating real-world problem of online movie streaming. While the customers' exact demands are of course not known perfectly ahead of time, there is almost always some indicator as to how popular a movie will be. We can expect a brand new and successful blockbuster to be in much higher demand than, say, a relatively unknown box office flop.

Hence a related problem that is of interest is to design a data layout given some demand distribution estimate. For instance, we might assume that the demand for each item follows a Gaussian distribution with a known mean and variance. Consequently, relaxing the requirement that each item is copied exactly ℓ times would allow the highest-demand items to be spread over a larger number of disks, potentially improving the maximum load.

This problem would likely respond well to an analysis similar to that which was presented in this work for the problem with unknown demands.

Maximum coverage objective

The area of this study having the greatest potential for improvement is that of the maximum coverage problem. We provided a brief initial estimation of the coverage of the uniform random data layout by deriving it directly from the maximum load estimate, which was the main focus of this work; however, the results would readily benefit from a more direct analysis. Rather than estimate the maximum load first, and then the maximum coverage, a direct analysis—perhaps using the same probabilistic techniques employed in our maximum load estimates—could give a better estimate.

An immediate improvement that could be made would be to generate additional curves for Figure 7.8 on page 54 for using a larger number of parts in the maximum load estimate. We saw that doing so yielded improvements for the maximum load, so there is no reason to suspect that doing so would not yield similar gains in the coverage objective. With more time, or more computing resources, these tests could be run to provide a better estimate of the coverage.

Approximation ratios

In section 6.3 we began a preliminary analysis of the approximation ratio Δ of the worst-case maximum load for the uniform random data layout, which suggested that $\Delta \approx 1.4$. Testing this for additional problem instances would do much to increase the confidence we can have in this result.

Ideally, we would like to find a concrete value for Δ such that no matter which demand distribution is used, the worst-case bound we get for the maximum load is no more than Δ-times the optimum maximum load. Finding a good bound for Δ would guarantee the good performance we already suspect from the uniform random data layout. One possible approach could be to modify MIP 1 to try to maximise the ratio Δ by allowing the demand distribution to be variable. This would find the worst-case demand distribution such that the maximum load estimated via our technique is high, but the optimum is low. As it stands, our maximum load estimate does not lend itself towards this kind of treatment. Hence, further research into reformulating an estimate that is more amenable to this kind of usage in linear programs could have applications in this area.

Furthermore, methods to speed up the time taken to solve MIP 1 and MIP 2 would be useful. Currently, MIP 1 is almost prohibitively slow, taking roughly one hour to solve one of the standard problem instances. The obvious linear relaxation (allowing fractional solution whereby an item can be partially assigned to a storage disk, i.e. allowing $x_{uv} \in [0, 1]$ instead of $x_{uv} \in \{0, 1\}$ for all $(u, v) \in E$) will always return a maximum load of $\bar{L}$, since every item can then be spread onto every disk. Hence other techniques to either speed up the integer program or to find a good approximation would be helpful.

Simulations and experiments

Our current estimate of the maximum load obtained from using the uniform random data layout is very good, and fairly close to optimal. Some interesting further research could be to run a series of simulations on theoretical demand distributions (like the Zipf distribution) to determine how well our estimate matches up with the real expectations.

Taking this a step further would be to stop guessing what the demand patterns for this domain look like, and actually run experiments using real-world client demand. This would help to pin down exactly what kinds of distributions we expect to see, which would allow more accurate estimations of the kind of performance that can be expected.

CHAPTER 9

Conclusion

In this work, we have presented the uniform data layout problem with unknown demands, and identified two possible objectives associated with it: minimising the maximum load across the storage disks (the load balancing objective), and maximising the coverage of the client demand (the maximum coverage objective). To this end, we proposed the uniform random data layout and established a framework for the analysis of the maximum load that we can expect to attain.

The upper bound on the maximum load found from this analysis is in fact a demand-assignment agnostic bound of the maximum load. That is, it allows us to say that, for all possible permutations of item demand assignments, the maximum load is less than our lower bound with high probability. This is a highly useful result, since is guarantees a worst-case performance even when it is not known which items will be the most popular. This is a likely scenario that could occur in a real-world setting; the general shape of the demand distribution is known, yet which specific items are high-demand is not known, or might change over time.

Using this analysis, we evaluated the uniform random layout on a series of large problem instances, with item demands governed by the Zipf distribution. By adjusting some parameters associated with the distribution (the Zipf exponent s and offset q) we were able to mimic other (less skewed) distributions. We found that our data layout is fairly robust to changes in parameters, and outperforms a naïve deterministic baseline significantly.

A more refined analysis was then conducted, which led to an algorithm for computing the maximum load estimate given a partition of the demands. This algorithm was seen to run in $\mathcal{O}(m^{t+1})$ time, where t is the number of parts used and m is the number of data items. Two methods for choosing an appropriate partition were presented: the ratio heuristic, which very quickly decides upon a good demand partition; and the linear search, which more slowly finds a significantly better partition. This new approach demonstrated even greater performance from the uniform random data layout on the load balancing objective.

By designing a mixed integer program, we were able to solve a number of test problem instances, which gave a bound on the performance we could expect from any data layout. This

showed that the uniform random data layout is near optimal, and appears to be a 1.4-approximation algorithm (on the various test instances considered).

We derived an estimate for the maximum coverage objective based on the maximum load estimates already found. Two additional methods were considered, which improved the analysis: one based on excluding the highest-demand items from consideration, and the other on truncating those demands. The latter was seen to give much better estimates, yet still fell well short of the optimum. The upper bound on the performance was determined using another mixed integer program for solving the maximum coverage data layout problem. The bounds obtained were nowhere near this optimum, so we conducted a series of simulations to generate random data layouts and find the maximum coverage for each. From this it became clear that the method by which we derived the maximum coverage estimate was not sufficiently tight to demonstrate a good level of performance.

Hence we found that the uniform random data layout is a very good solution for the load balancing data layout problem, and were able to provide a means by which it can be evaluated on any given demand distribution in order to determine it's viability to the specific problem domain. Additionally, it was also seen to achieve very good results for the maximum coverage objective, with a strong possibility of improvements to this estimate given further research.

Bibliography

[And01] Eric Anderson et al. 'An experimental study of data migration algorithms'. In: *Algorithm Engineering*. Springer, 2001, pp. 145–158.

[Ber11] Petra Berenbrink et al. 'Randomized diffusion for indivisible loads'. In: *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2011, pp. 429–439.

[Ber94] Steven Berson et al. *Staggered striping in multimedia information systems*. Vol. 23. 2. ACM, 1994.

[CK93] Tzi-cker Chiueh and Randy H Katz. 'Multi-resolution video representation for parallel disk arrays'. In: *Proceedings of the first ACM international conference on Multimedia*. ACM. 1993, pp. 401–409.

[ELZ86] Derek L Eager, Edward D Lazowska and John Zahorjan. 'Adaptive load sharing in homogeneous distributed systems'. In: *Software Engineering, IEEE Transactions on* 5 (1986), pp. 662–675.

[FS09] Tobias Friedrich and Thomas Sauerwald. 'Near-perfect load balancing by randomized rounding'. In: *Proceedings of the 41st annual ACM symposium on Theory of computing*. ACM. 2009, pp. 121–130.

[Gan08] Sumit Ganguly. 'Data stream algorithms via expander graphs'. In: *Algorithms and Computation* (2008), pp. 52–63.

[Gol00] Leana Golubchik et al. 'Approximation algorithms for data placement on parallel disks'. In: *ACM Transactions on Algorithms* 5.4 (Oct. 2000), pp. 1–26.

[Gol06] Leana Golubchik et al. 'Data migration on parallel disks: Algorithms and evaluation'. In: *Algorithmica* 45.1 (May 2006), pp. 137–158.

[HLW06] Shlomo Hoory, Nathan Linial and Avi Wigderson. 'Expander graphs and their applications'. In: *Bulletin of the American Mathematical Society* 43.4 (2006), pp. 439–562.

[Hal01] Joseph Hall et al. 'On algorithms for efficient data migration'. In: *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2001, pp. 620–629.

[KK06] Srinivas Kashyap and Samir Khuller. ‘Algorithms for non-uniform size data placement on parallel disks’. In: *Journal of Algorithms* 60.2 (Aug. 2006), pp. 144–167.

[KKM06] Samir Khuller, Yoo-Ah Kim and Azarakhsh Malekian. ‘Improved Algorithms for Data Migration’. In: *Proceedings of the 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*. 2006, pp. 164–175.

[KKW03] Samir Khuller, Yoo-Ah Kim and Yung-Chun Justin Wan. ‘Algorithms for data migration with cloning’. In: *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM. 2003, pp. 27–36.

[Lub12] Alex Lubotzky. ‘Expander graphs in pure and applied mathematics’. In: *Bull. Amer. Math. Soc* 49.113-162 (2012), p. 45.

[MGS98] S Muthukrishnan, Bhaskar Ghosh and Martin H Schultz. ‘First-and second-order diffusive methods for rapid, coarse, distributed load balancing’. In: *Theory of computing systems* 31.4 (1998), pp. 331–354.

[Mar73] G A Margulis. ‘Explicit constructions of expanders’. In: *Problemy Peredači Informacii* 9.4 (1973), pp. 71–80.

[Mit96] Michael David Mitzenmacher. ‘The power of two choices in randomized load balancing’. PhD thesis. Citeseer, 1996.

[ORS96] Banu Ozden, Rajeev Rastogi and Abraham Silberschatz. ‘Disk striping in video server environments’. In: *Multimedia Computing and Systems, 1996., Proceedings of the Third IEEE International Conference on*. IEEE. 1996, pp. 580–589.

[Opt13] Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2013. URL: http://www.gurobi.com.

[RMS01] Andrea W Richa, M Mitzenmacher and R Sitaraman. ‘The power of two random choices: A survey of techniques and results’. In: *Combinatorial Optimization* 9 (2001), pp. 255–304.

[RSW98] Yuval Rabani, Alistair Sinclair and Rolf Wanka. ‘Local divergence of Markov chains and the analysis of iterative load-balancing schemes’. In: *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*. IEEE. 1998, pp. 694–703.

[SMRN00] Jose Renato Santos, Richard R Muntz and Berthier Ribeiro-Neto. ‘Comparing random data allocation and data striping in multimedia servers’. In: *ACM Sigmetrics Performance Evaluation Review*. Vol. 28. 1. ACM. 2000, pp. 44–55.

[SS12] Thomas Sauerwald and He Sun. ‘Tight Bounds for Randomized Load Balancing on Arbitrary Network Topologies’. In: *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*. 2012, pp. 341–350.

[ST01] Hadas Shachnai and Tami Tamir. ‘Polynomial time approximation schemes for class-constrained packing problems’. In: *Journal of Scheduling* 4.6 (2001), pp. 313–338.

[SV97] Prashant J Shenoy and Harrick M Vin. ‘Efficient striping techniques for multimedia file servers’. In: *Network and Operating System Support for Digital Audio and Video, 1997., Proceedings of the IEEE 7th International Workshop on*. IEEE. 1997, pp. 25–36.

[SnT01] Hadas Shachnai and Tami Tamir. ‘On two class-constrained versions of the multiple knapsack problem’. In: *Algorithmica* 29.3 (2001), pp. 442–467.