Analyzing BANG and NIBGF files using the *Capitalist* model (Draft)

Stephen Taylor, Nabil Hachem, and Stanley Selkow
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609, USA
email: {staylor,hachem,sms}@cs.wpi.edu
phone: (508) 831-{5409,5669,5449}
Fax: (508) 831-5776

Abstract

A modeling technique, the *Capitalist* model, is outlined for analyzing multi-dimensional file structures. It is particularly appropriate where the data keys may have non-uniform data distributions. The basis of the model is to assume that the current distribution of data in an existing file is a good predictor of the underlying "true" distribution.

The method is applied to formal analysis of the BANG and NIBGF directory structures. The average cost for search and insertion is found to be logarithmic in the file size. The order constant is quite small and depends on the capacity of a bucket. Simulation confirms the analytic results. This first use of the *Capitalist* model suggests its usefulness to the analysis of other multidimensional file structures.

1 Introduction

Multi-dimensional bucket files are data structures for storing records with many fields or ***keys***. A bucket will contain records which are relatively close to one another if the records are regarded as points in metric space. A variety of such structures have been proposed, among them the K-d-tree [Ben75], the Quadtree [FB74], the Grid File [NHS84], the h-B tree [LS90], and the two structures discussed in this paper, the BANG file [Fre87] and the NIBGF [OM91].

Our particular interest is in comparing file structures and algorithms with formal mathematical analysis. Analytic models permit asymptotic comparisons between algorithms with a nicety unachievable by performance monitoring or simulation. A recurring problem of formal analysis is to provide mathematically tractable models of structures. Commonly used modeling techniques include: ***Fractional Progress***, the analysis of a file in terms of percentage of transition from one mode to another [Lar85, HB92]; this fades into ***Demographics***, which studies the populations in various modes [Yao79, Lom81]; and ***Geometric Transformation***, which makes the structure under study similar or equivalent to some other, well-known one [LS90, OM92]. Modeling simplifications include studying a structure only as it grows, or only at an hypothesized ***steady state.*** Especially relevant to us are assumptions about the data distribution; the simplest assumption is that data is uniformly distributed. More general is our ***Capitalist*** model: 'the rich get richer,' or 'data attracts more data.'

The contribution of this paper is two-fold: we introduce the ***Capitalist*** model as an addition to the model-builders toolkit, and we apply it to an average-case analysis of the BANG/NIBGF directory scheme. We find that the average height of a node in the directory tree is logarithmic in the number of buckets.

In Section 2 we introduce the ***Capitalist*** model of file growth as an aid to analyzing file structures. It is contrasted with the Bernoulli and Poisson models developed by [FNPS79, Reg85, Fla83]. In Section 3 we draw upon the file description from [OM92], recasting the notation somewhat. We analyze data bucket splits and apply one variant of the ***Capitalist*** model to obtain a probability distribution for the directory tree height in the BANG file. Finally we discuss performance implications with respect to NIBGF. Section 6 compares the results of the analysis with simulations. We conclude and summarize in Section 7.

2 The Capitalist Method

The ***Capitalist*** model of file growth is particularly apt for non-uniform data distributions. When the underlying data structure is appropriate for this model, we posit that density of current data values in the file structure corresponds to variations in density in the unknown underlying distribution of the data in the domain.

There are two other important techniques for modeling data distributions in multi-dimensional file structures. Under the ***Bernoulli*** model, the record keys are assumed to be uniformly distributed throughout the keyspace[1]. The probability of insertion into a bucket is the fraction of the entire keyspace which it covers, and the probability of various numbers of records in a bucket may be modeled with the binomial distribution. Under the Bernoulli model, the populations of the buckets are not independent.

When we want the probabilities of the number of keys in each bucket to be independent, we can use the ***Poisson*** model. In this model the number of records in the file has the Poisson distribution, and the probability distributions for the buckets are independent, and have Poisson distributions.

[1] Uniformity is a property of the *model* as introduced by [FNPS79], not of a Bernoulli or binomial distribution. Clearly each bucket could have a Bernoulli distribution with a different probability p_b.

The problem with both of these models is that useful analysis using them assumes a uniform distribution over a finite keyspace. This is not a realistic assumption for real-world datasets, and several of the multi-dimensional file structures anticipate non-linear transformations on the keyspace, which can be expected to lead to non-uniform distributions of data.

In contrast, the ***Capitalist*** model deals easily with non-uniform distributions. The assumption of the ***Capitalist*** model is that past behavior will be a good predictor of future behavior. Therefore, buckets which cover areas that have had equal numbers of previous insertions are expected to have equal probability of an insertion, independent of the total area of the key-domain which they cover.

Clearly the model is not applicable to all file structures. For example, in Linear Hashing [Lit80], full buckets lead into overflow chains. So the maximum number of records in a bucket does not correspond to a ceiling probability for a bucket hit. (However, the number of records in a bucket and its overflow chains may be a good estimator of the expected value of another hashing collision.) In some applications insertion history may not be a good predictor of reference history, although intuitively we might expect insertion history to be as good an estimator of references as of insertions.

We outline the basis for two variants of this model: the per-key and the per-bucket approximations.

Per-Key approximation

The per-Key approximation is a straightforward form of the ***Capitalist*** principle. A version of the per-Key approximation was first used by [Yao79] in his fringe analysis of the B-tree. Each key in the file is assumed to stake a claim to a portion of the dataspace. Where keys are sparse, the size of a claim is large; where keys are dense, the size of the interval surrounding a key will be small. Each interval is assumed to be equally likely to receive an insertion, independent of its size.

Per-Bucket approximation

Depending on the access method, there may be various numbers of keys stored in each bucket. For example, data buckets in B-trees or B^+-trees are always at least half full [Com79] (unless the file has only one bucket.) BANG files have data buckets which range in utilization from one-third to entirely full.

The per-Bucket approximation ignores these variations. It assumes that each bucket is equally likely to receive new insertions, and ultimately to split, independent of its current utilization. Obviously this is a simplification, but the simplification makes the mathematics easier. Furthermore, for non-uniformly distributed data, the areas covered by buckets are likely to be much better approximations to areas of equal record density than other ***a priori*** estimates like equal area.

Related Work

The ***Capitalist*** model differs from [Yao79] and related work by Baeza-Yates in that

- Although Yao formulated the idea of equally-probable intervals, and used it in the analysis of [Yao79], he did not examine it in detail; the bulk of that analysis deals with transformations between ““fringe” subtrees.
- The ***Capitalist*** model tracks the statistic “keys per bucket” rather than its dual, “data-space per key.”
- It unifies the “ideal hashing” of [BY89] and “equally probable key-intervals” in a single concept.

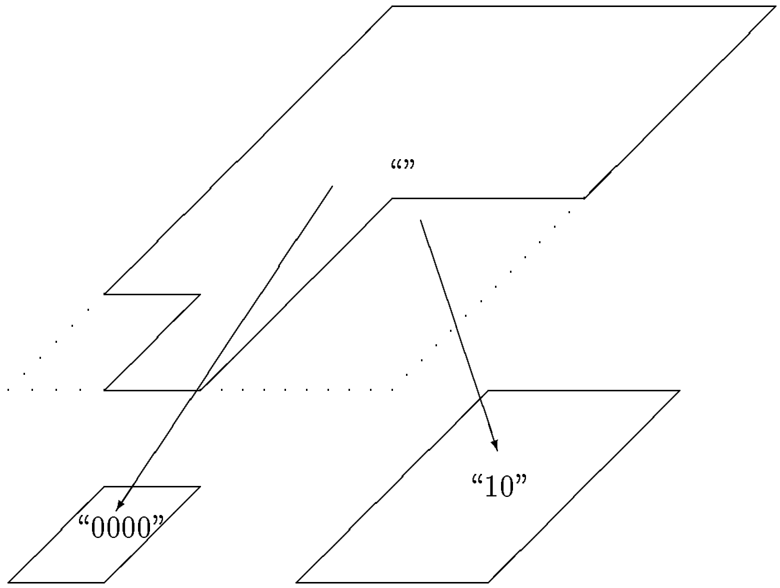

Figure 1: space covered by the data buckets in a BANG directory tree

- The intuitive foundation in the key-distribution is explicit, rather than implicit.
- Most importantly, founding the model in the key-distribution gives a basis for analysis of the model itself.

3 Analysis of the BANG Directory Tree

In this section, we apply the ***Capitalist*** technique to the BANG file. We start by a description of the BANG directory tree, followed by an analysis of the split probabilities of data buckets in the file. We then provide a detailed analysis and estimate of the directory height of the BANG file. The structure we analyze is the abstract directory structure, for which a number of different physical implementations are possible. One subset of the possible physical implementations include NIBGF directories. We postpone discussion of the physical implementation until section 4, where we compare BANG and NIBGF.

3.1 The BANG Directory Tree

A BANG file [Fre87, OM91] contains k-field records which are each mapped to a point in $[0,1]^k$. The file has a directory tree or forest of trees which has one node for each data bucket in the file. Each node in the directory tree has an associated subspace descriptor string, which denotes the canonical bounding subspace for the node and all its descendant nodes in the directory. The bounding subspaces of sibling nodes do not overlap, but the bounding subspaces of parent and child nodes always overlap.

Possible bounding subspaces are created by successive halving of $[0,1]^k$ into subspaces, such that a string of binary digits can be assigned to every rectangular subspace, e.g. “0011” or “001”. Figure 2 illustrates a few of the canonical subspaces and their descriptors for a two-dimensional space. One subspace completely contains another if its descriptor string is a prefix of the second subspace's string. A data point may be given a (potentially infinite) descriptor also, such that the descriptor string of every subspace which contains it

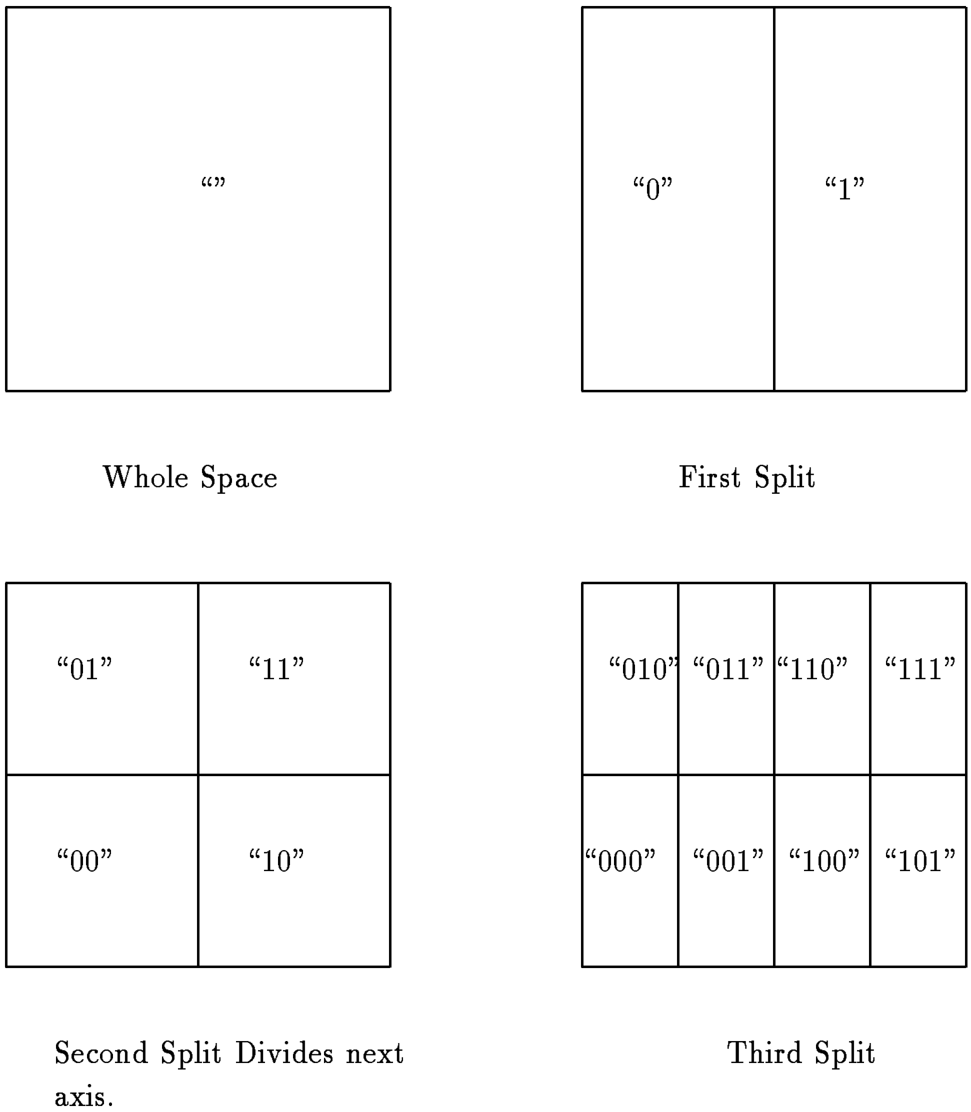

Figure 2: The canonical subspace splits in a BANG directory tree

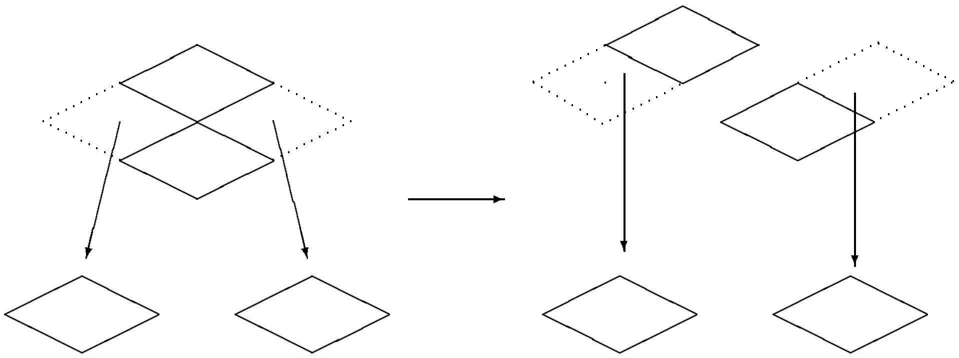

Figure 3: Bucket split creates two peers

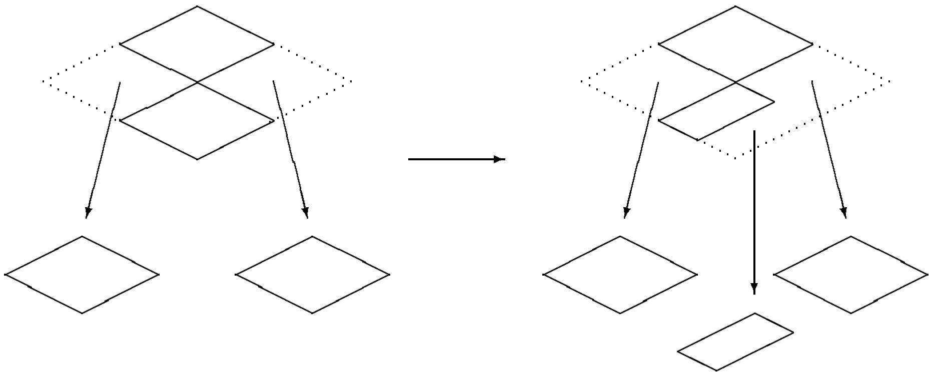

Figure 4: Bucket split creates descendant bucket

is a prefix. A descriptor string for a data point can be generated by a perfect shuffle, or ***Z-ordering*** [Ore83] of the bit values of its k coordinates into a binary string.

In Figure 1, which illustrates a two-dimensional file with three data buckets, the root node subspace descriptor would be the empty string "", corresponding to the entire unit square. This is the area shown in the figure with a dashed line, and is the area covered by this node and all of its descendants. However, the data bucket corresponding to the node contains only those points not covered by some descendant. This is the gnomon drawn with solid lines. The two children of the top node have subspace descriptors "10" and "0000".

A point is found or inserted in the bucket at the outermost node in the directory tree which contains it. The cost of a search in this scheme depends on the depth of the directory and the fanout of the nodes. The size of the directory is directly proportional to the number of data buckets, so it remains only to see how the other parameters relate to the size.

3.2 Data Bucket Splits

A first step in modeling the behavior of the directory tree as it grows is to describe the behavior of a single bucket when it overflows as the result of insertions. There are two ways to maintain the properties of the data structure: we can split the bucket somewhat evenly, replacing a single node in the directory tree with two nodes (Figure 3); or we can carve a rectangular subspace out, and make it a descendant in the the tree (Figure 4). Which of these two occurs depends on how the data is distributed over the keyspace spanned

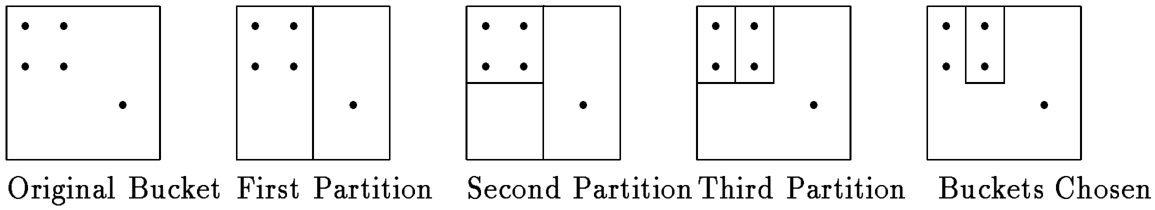

Figure 5: Intermediate steps in bucket partition process

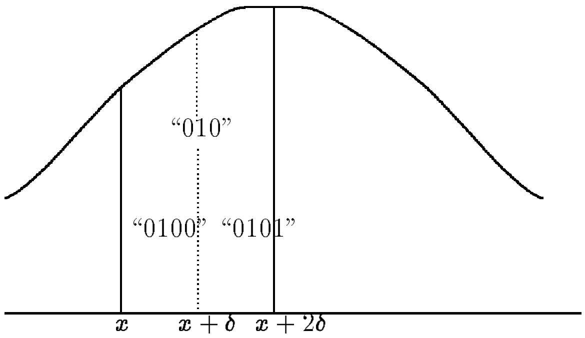

Figure 6: Probability density of data may vary across bucket

by the bucket.

If the first partition of the bucket succeeds in dividing the records so that neither part contains more than twice as many records as the other, we call this a ***Peer Split.*** If not, then successive partitions are considered until the utilization constraint is met. The resulting division of data between buckets will require a child in the directory tree. This is referred to as a ***Subspace Split.*** The 1 :: 2 constraint permits use of a greedy partition algorithm which always subpartitions the fuller of the two parts created at the last (sub-)partition. Figure 5 illustrates the progress of the greedy partition algorithm for a two-dimensional file and a bucket capacity of four.

The height of the directory tree is related to the split behavior; if every bucket overflow results in a peer split, the height of the directory will be zero. The only way the directory can become deeper is through a subspace split. Thus it is important to determine the probability p_S that a split will be a subspace split.

3.3 Probability of a Subspace Split

To model the split behavior, we note that there are two sources for variation in the probability density across a node. First, there is the gradient in the underlying probability distribution of the data. For example, consider the imaginary case illustrated in Figure 6. This figure shows a variation in marginal probability density for x in different parts of the key space, including a variation across the bucket "010". When we partition bucket "010" we expect to find that fewer of the datapoints will fall into partition "0100" than "0101", because the relative cumulative marginal densities p, for bucket "0100" is smaller than the cumulative density q, for bucket "0101". Given p and a bucket capacity $b_{\max}$, we can predict the probability of any population m of bucket "0100" at the time of a split, when we know that there are $b_{\max} + 1$ datapoints in bucket "010". Using the binomial distribution, this probability is

$$\binom{b_{\max}+1}{m} p^m (1-p)^{b_{\max}+1-m}$$

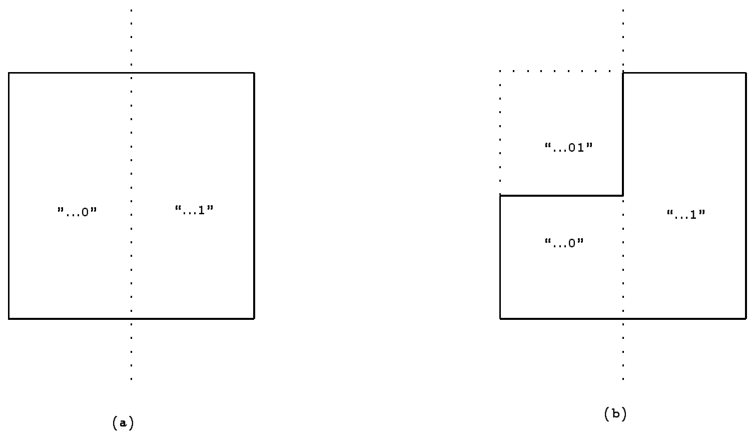

Figure 7: Space claimed by descendants changes shape of bucket

The probability, p_S, of a subspace split (one which generates a descendant in the directory tree, or equivalently, one requiring more than one partition) is

$$p_S = \sum_{\substack{0 \le m < \frac{b_{\max}+1}{3} \\ \frac{2}{3}(b_{\max}+1) < m \le b_{\max}+1}} \binom{b_{\max}+1}{m} p^m (1-p)^{b_{\max}+1-m} \tag{1}$$

that is, the probability that a second partition will be necessary to maintain storage utilization is the sum of two tails of the binomial distribution.

3.3.1 Digression on Binomial tails

If $b_{\max}$ is large and p is central, the tails are small and far from the probability mass centered at the mean, $b_{\max} \cdot p$, and we can neglect the summation, and consider only the probabilities p and q. The expected populations $\overline{p}$ (or $\overline{q}$) of the two pieces are binomially distributed random variables with mean $(b_{\max}+1)p$ (or $(b_{\max}+1)q$) and variance $\sigma^2 = (b_{\max}+1)pq$. We can use the Normal approximation to the Binomial distribution to compute a 95% confidence interval for $\overline{p}$

$$\begin{aligned} \overline{p} &= (b_{\max}+1)p \pm \Phi^{-1}(.975)\sqrt{(b_{\max}+1)pq} \\ &= (b_{\max}+1)p \pm (1.96)\sqrt{(b_{\max}+1)pq} \quad \text{with 95\% confidence} \end{aligned}$$

Since $\sqrt{p(1-p)}$ has its maximum value at $p = \frac{1}{2}$, we have

$$\begin{aligned} \overline{p} &= (b_{\max}+1)p \pm \sqrt{(b_{\max}+1)} \\ \overline{p} &= (b_{\max}+1)p(1 \pm \frac{1}{p\sqrt{(b_{\max}+1)}}) \quad \text{with 95\% confidence or better} \end{aligned}$$

and the error $(1 \pm \frac{1}{p\sqrt{(b_{\max}+1)}})$ approaches 1 as $b_{\max}$ increases. We'd like $\frac{1}{p\sqrt{(b_{\max}+1)}} \ll p$. We're particularly interested in values of p in the neighborhood of $\frac{1}{3}$, because the BANG algorithms treat splits in the range $\frac{1}{3} < \frac{\overline{p}}{b_{\max}+1} < \frac{2}{3}$ specially. We can solve for the value of $b_{\max}$ which brings the error term less than $p/2$ at $p = \frac{1}{3}$:

$$p/2 > \frac{1}{p\sqrt{(b_{\max}+1)}}$$

$$
\begin{aligned}
\frac{1}{6} &> \frac{1}{(1/3)\sqrt{(b_{\max}+1)}} \\
\frac{1}{18} &> \frac{1}{\sqrt{(b_{\max}+1)}} \\
18 &> \sqrt{(b_{\max}+1)} \\
324 &> (b_{\max}+1) \\
323 &> b_{\max}
\end{aligned}
$$

With plausible physical block sizes of 4096 bytes, this seems like a reasonable number of records to fit in a bucket.

3.3.2 Digression on continuity

If the underlying distribution of the data is continuous, then as the number of buckets in the file increases, and each bucket covers a smaller and smaller portion of the key-domain, we would expect that the cumulative probability in adjacent buckets of the same coverage would be about the same. In particular, we'd expect the probabilities of the two sub-buckets of a split to converge to $\frac{1}{2}$ as the area in the key-space which the bucket covers approaches zero.

However, there is nothing about the ***Capitalist*** model which requires continuity. Many real-world datasets concern finite populations, which are discontinuous at the level of individuals. For example, in an addressbook relation with schema $\{NAME, CITY, STATE\}$, we might find many entries for $CITY =$ 'Los Angeles'. The ***Capitalist*** model predicts that subsequent insertions will also include many entries for $CITY =$ 'Los Angeles'. Nearby regions of the key-space do not experience 'trickle-down.' There are no entries for 'Los Angelet' or 'Los Angeler'.

The file itself has a finite population, and so the representation of the underlying distribution provided by the file must always be discrete. We can, however, talk about apparent continuity. If the cumulative probabilities of adjacent areas of the keydomain tend to be about the same, then the data distribution is perceived to be continuous. If not,

- the distribution might be continuous if we considered smaller areas.
- the distribution might be discontinuous.

Capitalism claims that insertions into an area of the key-domain so far are an estimator of future insertions into the same area. It doesn't claim that insertions into one part of the area covered by a bucket predict insertions into other parts of the bucket.

In a continuous distribution, in the limit as bucket sizes decrease, probabilities of the respective halves of a split bucket converge to one-half. One could imagine a discontinuous function for which the probabilities converge to $p \neq (1-p)$; [Reg85] carries out an analysis of grid files for such a distribution function. In the antithesis of continuity, the relative probabilities of the halves of a split bucket might have any value. Of course $p + q = 1$, but p might have any value between 0 and 1. We call a distribution function with this property ***anti-continuous***.

Definition 1 *A distribution function $F(x)$ is* near-continuous *at x, δ if*

$$F(x+\delta) - F(x) \approx F(x+\delta) - F(x)$$

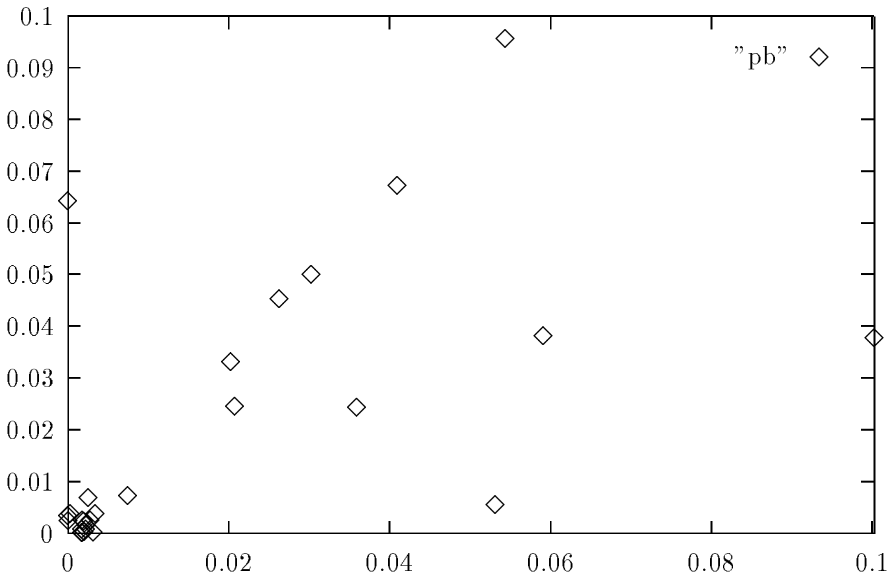

Figure 8: Successive probabilities of words

Definition 2 *A distribution function $F(x)$ is* non-continuous *at x, δ if*

$$F(x+\delta) - F(x) \not\approx F(x+\delta) - F(x)$$

A discontinuous function is non-continuous for every value of δ.

Definition 3 *A distribution function $F(x)$ is* anti-continuous *at δ if*

$$\frac{F(x+\delta) - F(x)}{F(x) - F(x-\delta)}$$

has a broad range of values.

Definition 4 *A distribution function $F(x)$ is* anti-continuous *at x if either of*

$$\frac{F(x) - F(x-\delta)}{\delta}$$

or

$$\frac{F(x+\delta) - F(x)}{\delta}$$

does not converge to a constant as δ goes to zero.

Anti-continuity is a stronger condition than discontinuity. However, it does not imply discontinuity; a continuous distribution function is anti-continuous over any interval on which the probability density is zero.

A dataset recording properties of individuals will be everywhere anti-continuous, although uncertainty about those properties can be used to introduce continuity by replacing point values with distributions.

Figure 8 plots against one another the absolute probabilities of words from an ordered German wordlist falling into successive intervals. For this plot, the interval size is one letter; the percentage of words

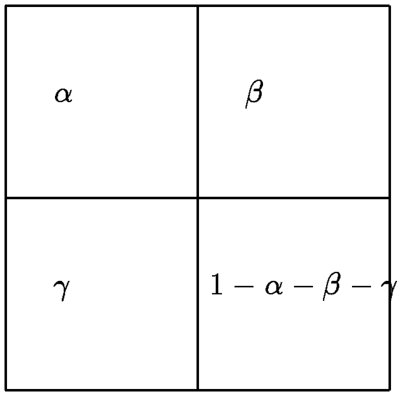

Figure 9: Assigning names to probability of insertion into bucket quadrants

beginnning with 'b' is plotted against the percentage of words beginning with 'c'; the percentage beginning with 'd' is plotted against 'e', and so forth. The plot has 26 points because upper- and lower-case are treated separately.

The graph exhibits both some near-continuous and some anti-continuous properties. We find a number of points whose abscissa and ordinate are both small. These correspond to the fact that relatively few words in the wordlist are capitalized, so the intervals in the region (A,Z) are all sparsely populated compared to the intervals in the region (a,z). This neighborliness is a near-continuous property. On the other hand, in all regions, the range of relative probabilities is broad. If the data distribution were continuous, we'd see the points clustered around the line $p = q$; instead they are widely distributed throughout the unit square.

A continuous distribution might exhibit this anti-continuous behavior for larger values of δ, and certainly the interval size for this plot is sufficiently large that it could still prove to be continuous for a smaller interval. But our intuition about word formation is that the contingent distributions of second letters are even less uniform than initial letters; if we divide the wordlist into 2^{72} intervals corresponding to the first nine letters of the words, almost all of the intervals will be empty, many will have only one word, but others will contain many words.

With this plausibility demonstration of anti-continuity in place, let us suggest a model for partition of buckets in the BANG file which is based on anti-continuity.

If the bucket size is big enough, the probability of a bucket split being a subspace split is the same as the probability that the relative probability of the first bucket division is less than $\frac{1}{3}$ or greater than $\frac{2}{3}$. That is,

$$p_S = P[p > \frac{2}{3}] + P[p < \frac{1}{3}]$$

Of course the assumption of anti-continuity does not tell us what the distribution of p is, but if every distribution is as likely as any other, we can make a plausible estimate of p_S based on the portion of the range of p which leads to a subspace split. That proportion is $\frac{2}{3} = .666\ldots$.

A refinement of this foolhardy estimate is also available. Suppose we plot the successive split probabilities of two axes of a BANG file against one another. Under an anti-continuous assumption, we'd expect the probabilities to be unrelated and have broad ranges.

Consider the four quadrants of a bucket which is splitting, as shown in Figure 9. The relative cumulative probability for each quadrant is shown as α, β, γ, $1 - \alpha - \beta - \gamma$. These probabilities are normalized to

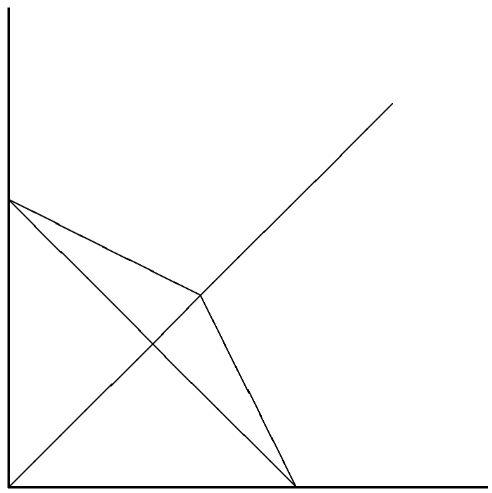

Figure 10: tetrahedron shows region of permissible quadrant probabilities

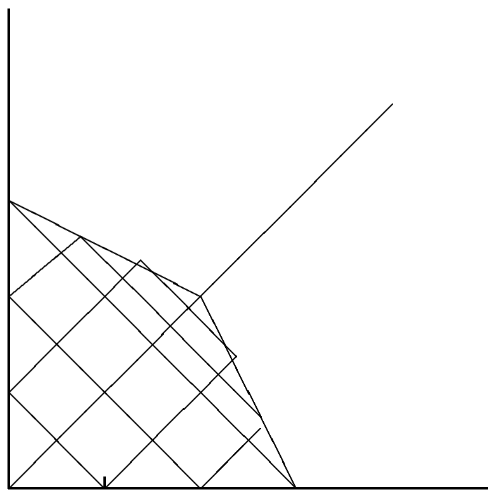

Figure 11: lsice shows values of α, β, γ for which bucket will split into two peers

sum to one; that is, we are for moment uninterested in the absolute probability of performing an insertion into the appropriate quadrant, but only the conditional probability given that an insertion strikes one of the four quadrants. Under an anti-continuous assumption, we treat α, β, and γ as ranging independently between 0 and 1. Once α, β, and γ are specified, we can plot the possible values as points in 3-dimensional space. Of course, α, β, and γ are not completely independent, since they must not sum to more than 1. Figure 10 shows the permissible positions in 3-space of the (α, β, γ) points as the volume of a tetrahedron. The tetrahedron reflects the constraints:

$$\begin{gathered} 0 \leq \alpha \leq 1 \\ 0 \leq \beta \leq 1 \\ 0 \leq \gamma \leq 1 \\ \alpha + \beta + \gamma = 1 \end{gathered}$$

Now consider the values of α, β, and γ which can result in a subspace split. These are those in Figure 11, and result from the added constraint

$$\alpha + \gamma < \frac{1}{3} \vee \alpha + \gamma > \frac{2}{3}$$

The volume of the entire tetrahedron of Figure 10 is 1/6. Working out the volume of the permissible region of Figure 11 we find it to be 17/162; dividing through by the volume of the tetrahedron, we find that the ratio of the volume of (α, β, γ) points which result in a subspace split to the possible volume of such points is 17/27 or .6296296...

This estimate of $p_S = .6296296\ldots$ should be tested by simulations on anti-capitalist datasets. Such simulations do not appear here.

3.4 A stochastic process for continuous distributions

We have a more complicated model for continuous data distributions For this case, the probability density over the two haves of the splitting bucket converge as the area covered by the bucket in the key space gets smaller, and difference in cumulative density of the two sub-buckets is due to the differing shapes of the subspaces covered by a bucket after spaces covered by descendant buckets are subtracted. Consider the two splits *(a)* and *(b)* in Figure (7). The bucket in Figure 7a has no descendants, and if the data were uniformly distributed, a new datum would be equally likely to fall into partition "...0" or "...1". The bucket in Figure 7b has a descendant, "...01". All of the datapoints in that quadrant of the keyspace are included in the descendant bucket. Therefore, if the data were uniformly distributed, a new datum would be twice as likely to fall into partition "...1" as "...0" (because "...0", having "...01" subtracted from it, is only half as big.) As we have seen in 3.3.2, these considerations for uniform distributions apply also to any continuous distribution in the limit as the number of buckets in the file increases and the area of the key-space spanned by each bucket decreases; and to 'near-continuous' distributions at appropriate bucket sizes.

Considering for the moment only *entire* buckets, those which have no descendants, and only uniform distributions, so that $p = q = \frac{1}{2}$, p_S tends to zero as $b_{\max}$ increases, and even for small values of $b_{\max}$ it is not large. Table (1) gives a few values of p_S for $p = \frac{1}{2}$.

Similarly, for a bucket covering a space like that of Figure 7b, if the data is uniformly distributed, the first partition will create regions "...0" and "...1". The two regions are nominally the same size, but "...0" has a child, and so insertions will have fallen into "...0" and "...1" in this bucket with probabilities $p = 1/3$ and $q = (1 - p) = 2/3$. The expected population of "...0" at the time of a split is $\frac{b_{\max}+1}{3}$ and in the limit

$b_{\max}$	p_S
3	.125
4	.375
5	.218...
6	.125
7	.289...
8	.179...
9	.109...
$\vdots$	
27	.036...
28	.061...
29	.043...
30	.029...
31	.050...

Table 1: Values of p_S for some values of $b_{\max}$ ($p = .5$)

as $b_{\max} \to \infty$ the probability of a subspace split is $p_S = 0.50$. For small $b_{\max}$, there is some variation; for example, for $b_{\max} = 5$, evaluating Equation (1) gives $p_S = 0.369$.

If we could estimate the proportion of buckets covering each of the possible subspace shapes, we could estimate an expected value for p_S. We consider fifteen possible shapes created by having or not having a descendant in each of four quadrants (Figure 12). These variations are grouped together according to how the descendant quadrants fall with respect to the bucket split line. The shape grouping named M, or '_ _' has no descendants on either side of the split line. The shape grouping named J, or '. _ _ _' has a single descendant quadrant on one side of the split. This is the shape illustrated in Figure 7b; there are four orientations possible. The shape C or '_ . _ .' has one descendant on each side of the split line; and the shape Z or '_ _ . .' has two descendants on one side, and zero or one descendants on the other side. The grouping consisting of a bucket which contains no datapoints, all of its space being subsumed by its descendants, cannot occur in a BANG directory.

This approach does not consider all possible subspace shapes. There are other possibilities: the partition shown in Figure 5 is an example of a directory node which is not in any of the groups M, J, C, or Z. In fact there an infinite number of these subspace shapes. We justify ignoring them with the numeric observation that, for a uniform data distribution, the probability of a subspace split of a bucket with no descendants creating a child smaller than a quadrant is quite small. This is about .02 for a bucket capacity of five and approaches 0 as this capacity increases.

For a bucket from group M, with $b_{\max} = 5$ and a uniform data distribution, we know from Table 1 that the probability of any subspace split occurring is $14/64 = .218875$. If we examine the probability of any particular quadrant becoming the single descendant as the result of a subspace split, we find that this outcome requires that its appropriate adjacent quadrant be full enough to require a second partition. For the particular case of $b_{\max} = 5$, this can happen if the desired quadrant has 3 datapoints, and its neighbor has 2 or 3, or the desired quadrant has 4 datapoints, and its neighbor has 1 or 2. The probability of one of these pairings (a, b) is described by the multinomial distribution

$$\binom{6}{a, b, 6-a-b} \left(\frac{1}{4}\right)^a \left(\frac{1}{4}\right)^b \left(\frac{1}{2}\right)^{6-a-b} = \frac{6!}{a!b!(6-a-b)!} \frac{1}{4^a 4^b 2^{6-a-b}} \tag{2}$$

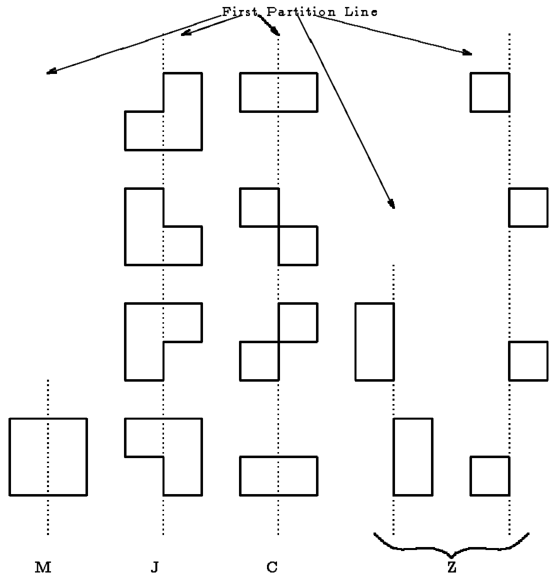

Figure 12: Space claimed by descendants changes shape of bucket

and since the pairings are mutually exclusive events, the probability that a particular quadrant will become a descendant is

$$\frac{2\binom{6}{3,2,1} + \binom{6}{3,3,0}/2 + 2\binom{6}{4,1,1} + \binom{6}{4,2,0}}{4096} = .05005 \tag{3}$$

We omit the case in which the quadrant contains 2 datapoints and the neighbor contains 3 or four, because the algorithm would choose the neighbor for the child in this case.

The case when both neighboring quadrants have three datapoints could result in either quadrant becoming the child; we need some care to avoid counting this case twice. As only one quadrant at most can become a descendant on a single split, the probability that some quadrant does so is $4(0.05005) = .2002$. Thus the probability that more than two partitions are necessary in this case is $.218875 - .2002 \approx .019$. While this probability is not vanishingly small, we may dare to hope that ignoring it will give us a reasonable approximation in the following.

As the result of an overflow, a bucket may be transformed in the following ways: (The numbers over the arrows indicate the probabilities of the transformation for a bucket capacity of five. The transition probabilities are slightly fudged to add to one for each shape. The same transformations occur, but the transition probabilities will be different for other values of $b_{\max}$. A value of $b_{\max} = 5$ was chosen for convenience of simulation. A general symbolic solution was not attempted.)

$$M \overset{.80}{\to} M'M' \tag{4}$$
$$\overset{.20}{\to} JM'' \tag{5}$$
$$J \overset{.66}{\to} Z'M' \tag{6}$$
$$\overset{.17}{\to} ZM'' \tag{7}$$
$$\overset{.17}{\to} CM'' \tag{8}$$
$$C \overset{.80}{\to} Z'Z' \tag{9}$$

$$\overset{.20}{\rightarrow} \quad ZM'' \tag{10}$$

$$Z \quad \rightarrow \quad ZM'' \tag{11}$$

Assuming that each variety of bucket is equally likely to overflow, and ignoring the $'$ marks which will be used later to indicate height in the directory tree, we can manipulate these transformations to get the Markov chain described by the following set of equations:

$$(n+1)\, m_{n+1} = nm_n + 0.8m_n + j_n + z_n + 0.2c_n \tag{12}$$

$$(n+1)\, j_{n+1} = nj_n + 0.2m_n - j_n \tag{13}$$

$$(n+1)\, z_{n+1} = nz_n + 0.83j_n + 1.8c_n \tag{14}$$

$$(n+1)\, c_{n+1} = nc_n + 0.17j_n - c_n \tag{15}$$

Each of the letters c_n, j_n, z_n, c_n indicates a proportion of the bucket population made up of the appropriate class of bucket when there are n buckets. Equation (12) can be read to say

> The number of buckets expected in class M after $n+1$ bucket overflows is the same as the number before the overflow, plus the expected number of buckets of class M created by the overflow. This number is .80 if the bucket which overflowed was of class M, 1 if the bucket which overflowed was of class J or Z, and .20 if it was a type C bucket.

Making the assumption that the stochastic process eventually converges to a steady state, we seek to find an approximation to the steady state by solving for a fixed point. We set $m = m_{n+1} = m_n$, $j = j_{n+1} = j_n$, etc., and include the condition $m + j + c + z = 1$. The resulting system of equations has the solution

$$m = .8286377196, z = .08145508784, c = .007043420617, j = .08286377196$$

We've computed the probabilities of subspace split for $p = .5$ (M and C) as .22; for $p = 1/3$ (J) as .35; and for buckets in class Z, there are no data points in one of initial partitions, so a partition must always generate a descendant, so $p_S = 1$. Thus we are able to compute an expected value for p_S (for a bucket size of 5) of

$$p_S \approx .22m + .35j + z + .22c \approx .29 \tag{16}$$

This value is quite close to the value we observe in the simulations reported in Section 6.

3.5 Directory height estimates

We now concentrate on estimating the directory height. A first approximation is made by assuming that each bucket is equally likely to split, and describing the state of the file as a vector showing the number of buckets at each depth of the directory.

This is a capitalist model of the file structure: it assumes that the subspaces spanned by buckets all have roughly equal probabilities of new insertions, even though they cover very different areas of key space; if they weren't deserving, they'd not have received insertions[2].

[2] **'To every one who has, will more be given.'** ***Matthew 25:29***

Generating function

Let us define:

p_S the probability that a split will be a ***subspace split***, that is, it divides the bucket-space unevenly, so that one of the resulting buckets covers a subspace of the other; the covering bucket will remain at its previous level in the directory tree, and the directory entry for the subspace bucket will be a child of the entry for the covering bucket.

p_P the probability that a split will be a ***peer split***, that is, that it will divide the bucket-space evenly to result in two peer buckets at the same level of the directory tree. Notice that $p_P + p_S = 1$.

$\pi_{l,n}$ the proportion of the population of buckets at level l when n buckets have been created.

We derive a generating function for the population distribution, making a number of simplifying assumptions about the system behavior:

- Every bucket is equally likely to receive the next key inserted. (The per-bucket approximation of the ***Capitalist*** model.)
- Every bucket is equally likely to split.
- p_P and p_S are constants, independent of the size of the file, insertion history, level of the splitting bucket in the tree, or distribution of the data.

Now we can characterize the population distribution by describing it with a generating function.

Theorem 1 *For each n the population distribution probabilities $\pi_{l,n}$ are described by the generating function:*

$$f_n(y) = \sum_{0 \le l < n} \pi_{l,n} y^l = \frac{1}{p_P + p_S y} \binom{(p_P + p_S y) + n - 1}{n} \tag{17}$$

We want to derive a generating function for the population distribution. We know a few of the boundary conditions.

$\pi_{l,0} = 0$		If there are no buckets in the population, there are none at any level.
$\pi_{0,1} = 1$		If there is only one bucket, it is at the root
$\pi_{1,2} = p_S/2$		The probability that the second bucket will generate a child is p_S,
$\pi_{0,2} = (1 + p_P)/2$		and there will then be two buckets in the tree.
$\sum_{l \ge 0} \pi_{l,n} = 1$		for each $n > 0$

We begin with a recurrence describing the growth of the directory tree when a bucket splits:

$$\begin{aligned}(n+1)\pi_{l,n+1} &= (n + p_P)\pi_{l,n}[0 \le l < n] + p_S \pi_{l-1,n}[0 < l \le n] \\ &\quad + 1 \cdot [l = 0][n = 0]\end{aligned} \tag{18}$$

This describes the change in the population of buckets at each level of the tree when the number of buckets increases from n to $n+1$. After the split, $(n+1)\pi_{l,n+1}$ is the expected number of buckets at level l. That expected value is the sum of the expected number of buckets before the split, $n\pi_{l,n}$,

the expected number of new buckets from a peer split at level l, $p_P\pi_{l,n}$

the expected number of new buckets from a subspace split at level $l-1$, $p_S\pi_{l-1,n}$

The expressions in square brackets are what [GKP88] calls 'Iverson's convention,' cited there as originating in [Ive62]. $[\boldsymbol{P}]$ represents the value 1 when $\boldsymbol{P}$ is true. When the conditions is not true, the value of the entire term, for example, $(n+p_P)\pi_{l,n}[0 \leq l < n]$ is 0. This convention keeps the boundary conditions visible during the generating function transforms which follow. The last term of (18) describes the directory tree's initial, empty state.

Define a family of generating functions (note that for $l \geq n$, $\pi_{l,n} = 0$):

$$f_n(y) = \sum_{0 \leq l < n} \pi_{l,n} y^l \tag{19}$$

Multiplying (18) by y^l and summing over non-negative values of l:

$$\begin{aligned}(n+1)\sum_{l\geq 0}\pi_{l,n+1}y^l &= (n+p_P)\sum_{l\geq 0}\pi_{l,n}y^l[0 \leq l < n] \\ &\quad + p_S\sum_{l\geq 0}\pi_{l-1,n}y^l[0 < l \leq n] + \sum_{l\geq 0}[l=0][n=0] \qquad (20)\\ &= (n+p_P)\sum_{l\geq 0}^{n-1}\pi_{l,n}y^l[n>0] \\ &\quad + p_S\sum_{l=1}^{n}\pi_{l-1,n}y^l[n>0] + [n=0] \qquad (21)\end{aligned}$$

substituting (19) appropriately:

$$(n+1)f_{n+1}(y) = (n+p_P)f_n(y)[n>0] + p_S y f_n(y)[n>0] + [n=0] \tag{22}$$

Multiply by x^n and sum (22) over all values of $n \geq 0$:

$$\begin{aligned}\sum_{n\geq 0}(n+1)f_{n+1}(y)x^n &= \sum_{n\geq 0}(n+p_P)f_n(y)[n>0]x^n + \sum_{n\geq 0}p_S y f_n(y)[n>0]x^n + \sum_{n\geq 0}[n=0] \\ &= \sum_{n\geq 1}nf_n(y)x^n + p_P\sum_{n\geq 1}f_n(y)x^n + \sum_{n\geq 1}p_S y f_n(y)x^n + 1\end{aligned}$$

define:

$$h(y,x) = \sum_{n\geq 1}f_n(y)x^n \tag{23}$$

then:

$$\frac{d}{dx}h(y,x) = \sum_{n\geq 1}nf_n(y)x^{n-1} \tag{24}$$

And a little substitution yields a differential equation.

$$\frac{d}{dx}h(y,x) = x\frac{d}{dx}h(y,x) + p_P h(y,x) + p_S y h(y,x) + 1$$

Doing a few cookbook manipulations:

$$\frac{d}{dx}h(y,x) + \frac{-(p_P + p_S y)}{(1-x)}h(y,x) = \frac{1}{1-x}$$

Multiplying through by $\exp(\int \frac{-(p_P+p_S y)}{(1-x)}dx) = (1-x)^{p_P+p_S y}$ and integrating yields:

$$\begin{aligned}(1-x)^{p_P+p_S y}h(y,x) &= C - \frac{(1-x)^{p_P+p_S y}}{p_P + p_S y} \\ h(y,x) &= \frac{C}{(1-x)^{p_P+p_S y}} - \frac{1}{p_P + p_S y}\end{aligned} \tag{25}$$

Where C is an as yet unknown constant introduced by the integration.

Expanding $(1-x)^{-p_P-p_S y}$ with the binomial theorem, we have

$$\begin{aligned}h(y,x) &= C\sum_{n\geq 0}\binom{-(p_P + p_S y)}{n}(-x)^n - \frac{1}{p_P + p_S y} \\ &= C\sum_{n\geq 0}\binom{(p_P + p_S y) + n - 1}{n}x^n - \frac{1}{p_P + p_S y}\end{aligned} \tag{26}$$

and, for $n > 0$,

$$f_n(y) = [x^n]h(y,x) = C\binom{(p_P + p_S y) + n - 1}{n} \tag{27}$$

We happen to know that $f_1(x) = 1$, so solving for C:

$$C = \frac{1}{p_P + p_S y} \tag{28}$$

and plugging this back into (27)

$$f_n(y) = \frac{1}{p_P + p_S y}\binom{(p_P + p_S y) + n - 1}{n} \tag{29}$$

What to do with a generating function

We can obtain any desired constant $\pi_{j,k}$ by repeated differentiation of $f_n(y)$, but this is not very satisfying. What Theorem 1 tells us is that $f_n(y)$ forms a probability generating function. This can be used to derive the expected value of the depth of a bucket when there are n buckets. This is expressed as $E_n(l) = \frac{d}{dy}f_n(1)$. Higher moments can also be easily obtained.

Making the attempt, we have[3]:

$$\begin{aligned} f_n(y) &= \frac{1}{p_P + p_S y}\binom{(p_P + p_S y) + n - 1}{n} \\ &= \frac{1}{p_P + p_S y}\frac{(p_P + p_S y + n - 1)^{\underline{n}}}{n!} \\ &= \frac{(p_P + p_S y + n - 1)^{\underline{n-1}}}{n!} \\ &= \frac{(p_P + p_S y + n - 1)!}{n!(p_P + p_S y)!} \\ &= \frac{\Gamma(p_P + p_S y + n)}{\Gamma(n+1)\Gamma(1 + p_P + p_S y)} \end{aligned} \tag{30}$$

utilizing `maple`, we get a derivative, which turns out to involve the ***digamma*** function,

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

$$\frac{d}{dy} f_n(y) = \frac{\Gamma(p_P + p_S\, y + n) p_S\ (\psi(p_P + p_S\, y + n) - \psi(1 + p_P + p_S\, y))}{\Gamma(n+1)\Gamma(1 + p_P + p_S\, y)} \tag{31}$$

Evaluating at $y = 1$ and noting that $p_P + p_S = 1$

$$\begin{aligned} f'_n(1) &= \frac{p_S\ (\psi(n+1) - \psi(2))}{\Gamma(2)} \qquad (32) \\ E_n(l) &= p_S\ (\psi(n+1) - 1 + \gamma) \end{aligned}$$

Referring to ([AS64],6.3.2, p.258), we learn

$$\psi(n+1) = \mathrm{H}_n - \gamma \tag{33}$$

So the asymptotic value of the expected depth of a node in the tree as n gets large is

$$E_n(l) = p_S \ln(n+1) - p_S \tag{34}$$

This value for the expected depth can be used to estimate average cost of algorithms which traverse the tree.

4 Comparing the NIBGF to BANG

NIBGF is a variant of BANG files in which the directory is prganized by ***levels*** in such a way as to permit a binary search between levels, and a binary search is possible within a level. The operations of search or insert are required to find the deepest node in the directory tree spanning a given portion of the key space. These operations can operate in $\lg h \cdot \lg w$ time; where h is the depth and w is the number of buckets of the

[3] **The falling factorial notation, used in Equation (30) may be unfamiliar.** $X^{\underline{n}} = X \cdot (X-1) \cdot \ldots (X-n+1) = \Pi_{0 \le i < n}(X-i)$

level under consideration. Based on our understanding of the expected height of a directory node, we can immediately predict that the binary search to determine directory level is not likely to be very valuable. In their analysis of search costs, [OM92] suggest that $\lg h$ is likely bounded by a small constant. We agree; our analysis shows that $\lg h = \lg(p_S \ln n)$. Just to pick a value likely to exceed the size of databases possible on today's machines, if $n = 2^{64}$ then $\ln n = 64(.6931\ldots) \approx 46$, and since $p_S < 1$, $\lg(46p_S) < 6$. (Actually our figure is the average depth; binary search cost depends on maximum depth, but it seems reasonable to estimate the order of search costs this way.)

Mean directory depth for many files will be small enough that a simple tree descent will be much less costly than a binary search between levels. Taking cost of a typical sequential search from [Knu73] as $5h/2 + 3$ and a typical binary search as $18 \lg h + 12$ (again these are for h representing maximum depth of the directory, not average) we can solve:

$$5h/2 + 3 \quad < \quad 18 \lg h + 12$$

A numerical solution gives $h < 42.6$ as the domain for which a sequential search is less expensive. Of course the costs of particular binary search algorithms may be different, and the break-even depth depends on the implementation.

Approximating the average depth as half the maximum depth, a maximum depth of 42 corresponds to a average depth of 21, which with a value of p_S of .29 can be used in the inequality

$$21 > .29 \ln n$$

to solve for the number of keys we'd need for binary search of levels to be a win. This gives a value of $n < 2.81 \cdot 10^{31}$ which rather handily exceeds the estimate of $2^{64} = 1.8 \cdot 10^{19}$ which we just offered as a reasonable maximum database size for present-day systems. If we use the value $p_S = .62$ estimated in section 3.3.2 for anti-continuous data distributions, we get a slightly more modest value of $n < 5.1 * 10^{14}$, probably within short-term hardware limitations, but still a very large database.

However, the issue of how to organize the descendants of a directory node has to be resolved, and the NIBGF structure is certainly a reasonable one. Since directory nodes may have thousands of children, an organization is required which permits rapid search through them. The NIBGF approach organizes all the nodes at a given depth of the tree, irrespective of their ancestry, into one ordered structure to which binary search may be applied. A tree walk involves performing a binary search at each level of the tree to find the descendants of the previous, higher node. There is consensus that the B-tree is an excellent structure for organizing this type of linearly sorted data. Abstractly and conceptually, each directory level is stored as a sorted array, but physically, it should probably be stored as a B-tree.

In our implementation, we permit the tree to become a forest, through peer splits of the root node. For large bucket sizes and uniform data distributions, the forest becomes a sorted array of data buckets, with a few layers of overflow levels. For non-uniform distributions, the number of levels needed to represent the data increases, but the total number of layers required remains relatively small.

The conceptual simplicity of this layered B-tree implementation is attractive. However, except for the root, most directory nodes will have relatively few descendants. The advantages of the layered scheme over an explicit pointer scheme are that:

1. Permits a binary search between levels, because the node, if any, covering a point at any level can be found by examining only the nodes of that level.

2. Peer splits do not require updating pointers in the parent or child nodes.

3. Fewer pointers are stored. The B-tree has internal pointers of its own, but these are probably shared among the children of several nodes.

Some of the disadvantages are:

1. A tree walk of the directory requires a binary search at each level.

2. Binary search touches directory pages remote from those pointing to interesting data.

3. There are hot-spots in the directory which are touched ***very*** frequently. For example, the highest and lowest levels of the tree are examined on every directory search.

As we have seen, the inter-level binary search advantage is actually no advantage at all. If we change the algorithms to omit use of inter-level binary search, we also mitigate disadvantages 1 and 2. Since search is incorporated into B-tree access, and one directory access to secondary storage suffices for B-trees of reasonable size, a superior, competing algorithm would have to make less than two accesses to follow its child pointers. In order to access to any child in a single disk reference, the descriptors and pointers for all the children would have to be stored in the parent node. This would increase the complexity and size of the node and the algorithms which manage it.

In summary, although the binary search capabilities of NIBGF are actually a liability, the data structure is a reasonably good implementation of the BANG file idea; more detailed comparison with other variants would require more implementation detail.

5 Storage utilization

The minimum storage utilization of a BANG file is 1/3. This can occur if there is a string of references which repeatedly splits a single bucket. The splitting algorithm of section 3.1 guarantees that the smallest new bucket created will be at least 1/3 full. If the added tuples are ordered so that the newly created buckets are only 1/3 full, and no bucket is ever referenced again after it is created, the maximum wasted storage will be achieved. In this case, the storage utilization of the BANG file will be 1/3. Sequential references to data buckets may be said to incur an overhead of 200%, because each tuple read will be accompanied by a transfer of twice as much empty space (the unused portion of the bucket.)

However, if we assume a less malignant distribution of new tuple insertions, buckets will continue to be referenced after they have been created. Then we can consider the utilization of a bucket over its lifetime. For each bucket, the least utilization it may have at creation time is 1/3; the maximum utilization at creation time is 2/3; and the mean value is 1/2. From the time a bucket is created until it splits again, its storage utilization will range from its creation value to 100%. The storage utilization of the whole file depends on the proportion of buckets for each possible value of the storage utilization.

The ***Capitalist*** model for storage utilization assumes that when a bucket is split into two new buckets, the relative utilization of the two buckets, that is, the number of tuples in each new bucket, determines the probability that subsequent tuples will be addressed to the region which a bucket spans.

For example, if a bucket is split which is currently receiving 20% of the tuples added to the file, and one of the resulting buckets is 1/3 full, the other 2/3 full, our model predicts that the first of the new buckets will receive 1/15 of subsequent tuples added to the file, and the second, or fuller, bucket will receive 2/15 of new tuples.[4]

[4] **'To every one who has, will more be given.'** *Matthew 25:29*

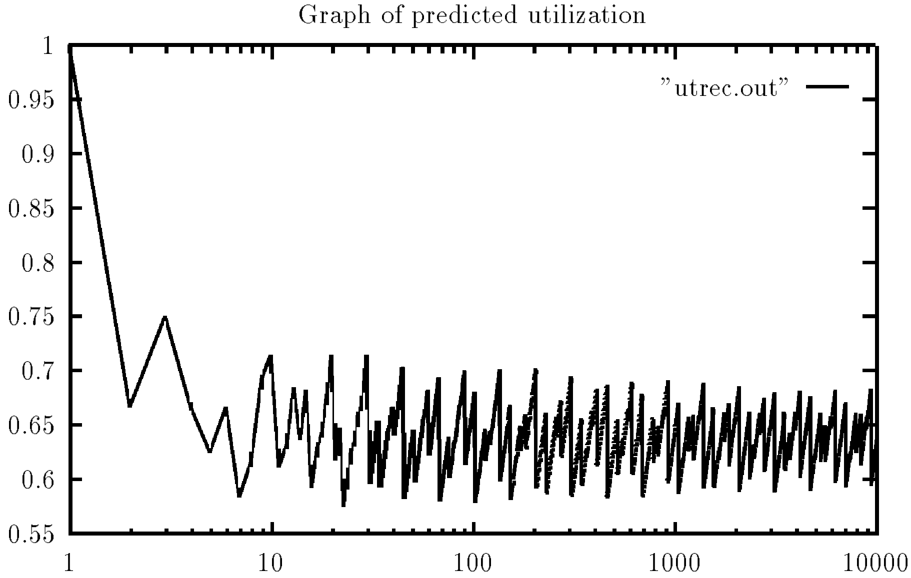

Figure 13: Utilization based on recurrence 35

If we make the added assumption that every split will be worst case, we can write a recurrence to approximate the number of buckets required to store xb tuples, where b is the number of tuples in a full bucket.

$$s(xb) = \begin{cases} 1 & \text{if } xb \leq b \\ s(\frac{xb-b}{3} + \frac{b}{3}) + s(\frac{2(xb-b)}{3} + \frac{2b}{3}) & \text{if } xb > b \end{cases} \tag{35}$$

That is to say, if the number of tuples on hand is less than one bucketful, then they will all fit into a single bucket. Otherwise they will be stored in more than one bucket. The file has a history of additions, and at some point it overflowed its first bucket and split into two sub-files. Since it was a worst-case split, one of the divisions included 1/3 of the then existing and subsequently added tuples, the other 2/3 of them. The subsequently added tuples are represented as $xb - b$ in the formulation of the recurrence; the previously added tuples appear as $\frac{b}{3}$ and $\frac{2b}{3}$. (Obvious algebraic simplifications are omitted.)

This recurrence gives a precise numerical answer, which obviously does not reflect the probabilistic nature of its assumptions. However, it seems like a reasonable model. The graph of figure(13) permits us to estimate the asymptotic utilization $\lim_{x\to\infty} \frac{xb}{s(xb)} = 0.64$.

[Lom81] provides a solution for steady-state file utilization when S of the data on a split goes to one bucket and $(1 - S)$ to another; that solution is given as

$$U = S \log\left(\frac{1}{S}\right) + (1 - S)\log\left(\frac{1}{1 - S}\right)$$

When $S = \frac{1}{2}$, this gives us the well-known utilization for a B-tree, $\log_e 2 = .693$, and when $S = \frac{1}{3}$ it gives an asymptotic utilization rate of 64.3%

We could attempt a mathematical analysis of equation (35), using Mellin transforms, but this would be only as effective as the original model, which is somewhat approximate. The graph suggests that the behavior of the utilization may have a log-periodic component, which is damped slowly, if at all.

Simulations appear to show a long-term log-periodic component in the utilization, but have a somewhat larger mean than predicted by the 1/3::2/3 model. (Nearer $.69 \approx \ln 2$) For very small bucket sizes, rounding

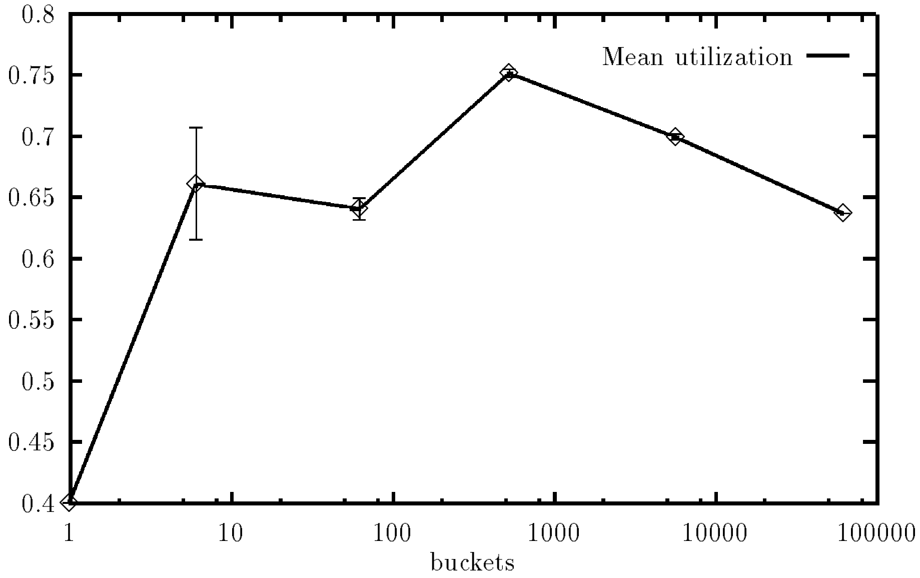

Figure 14: Simulated Utilization for bucketsize of 5

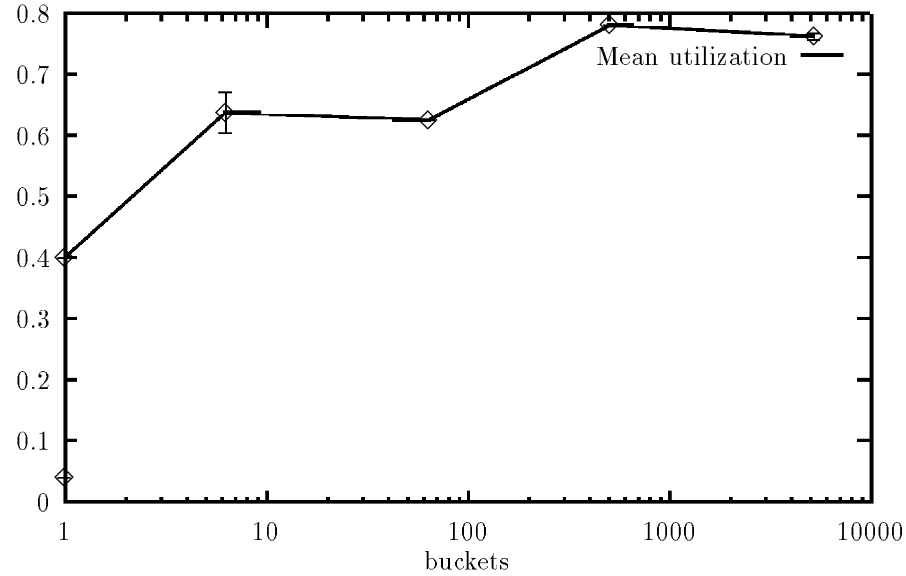

Figure 15: Simulated Utilization for bucketsize of 500

effects push the utilization up. For example, for a bucket size of 5, utilization varies from a low of 40%, rather than 1/3. For very large bucket sizes, (and uniform distributions), it will be be very improbable to put as few as 1/3 of the data points into a partition. Our graphs in figures (14) and (15) illustrate this.

6 Simulations

We coded a metered virtual-memory implementation of a two-dimensional NIBGF [OM92] variant of the BANG file [Fre87] and created files with large numbers of buckets, using uniform and normal distributions and bucket capacities of 5, 30, 60, 120 and 500 tuples. Statistics were collected on the fraction of bucket overflows which created a child in the directory, the storage utilization, and the height distribution of buckets in the directory tree. The graphs of Figure 16 and 21 show some of our results.

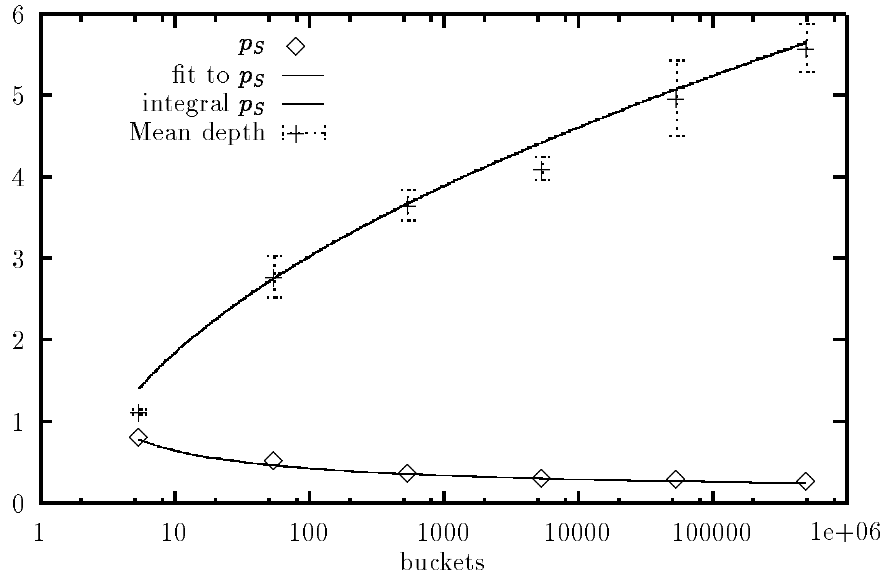

Figure 16: Simulation with normal distribution. Bucketsize = 5

For the simulation of Figure 16, the x and y coordinates of the two-dimensional data are independent Gaussian random variables with a mean of .25 and a standard deviation of .03125. This is a normal distribution centered in the lower left quadrant, as shown in Figure 17.

The simulation was performed with a bucket capacity of 5. Section 3.2 discusses the expected split behavior of this particular bucketsize at length.

In the graph we see p_S has a fairly large value when there are only a few buckets in the file. In fact, p_S is larger than our model of section 3.3.2 predicts for an anti-continuous data distribution. The first few splits must always generate descendants, because three-quadrants of the key-space contain essentially no data. However, as more buckets are inserted and the width of the key-space spanned by most buckets gets small relative to the standard deviation, the continuous nature of the distribution gradually asserts itself, and p_S asymptotically approaches .28. This is shown in more detail in Figure 18. This value is reasonably close to the approximate prediction of Section 3.2. Thus the simulation confirms the usefulness of the estimation techniques used in section 3.2.

Simultaneously, the slope of the average bucket depth tracks the change in p_S, as predicted by Equation

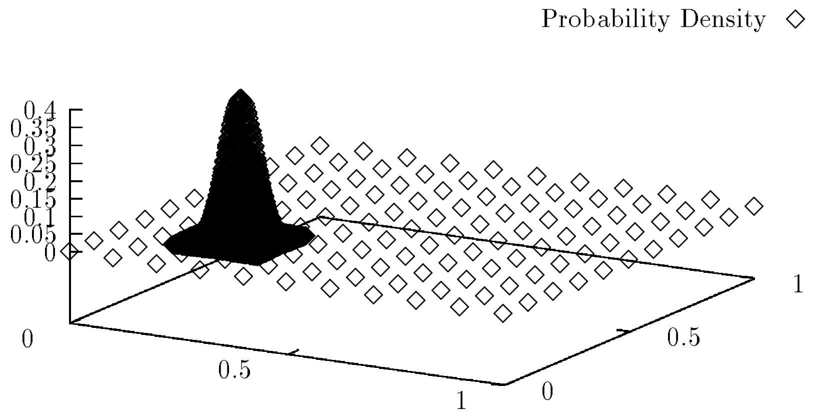

Figure 17: Simulated normal distribution is offset from center.

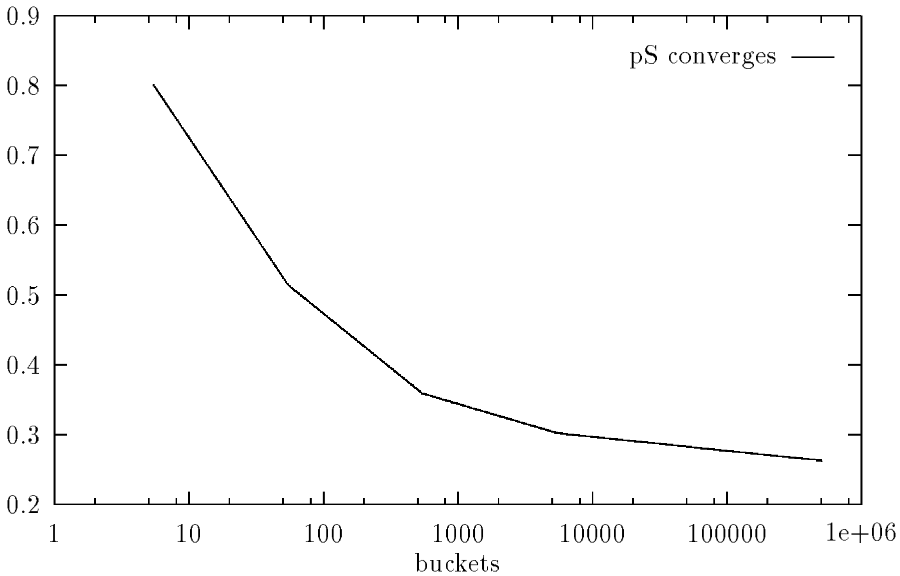

Figure 18: p_S detail. Normal distribution. Bucketsize = 5

(34). In Figure 16 a curve has been fit to the p_S observations, and the integral of the fitted curve is shown to track the observed stack depths.

In Figure 19 we see the the value of p_S tracks the predictions of section 3.2 very well. The average directory depth does not track the theoretical prediction nearly so well. This is probably because a uniform distribution is emphatically ***not Capitalist*** . When the first descendant bucket of a directory node is created, it covers at most one-third the area of its parent. For the uniform data distribution modeled by this simulation, the nodes nearest the leaves of the tree will receive fewer insertions than their parents, and split less often. In consequence we observe shallower trees than predicted by the assumptions which derived the theoretical depth.

There are a few features of the graphs which are not explained by the models. The most striking unexplained feature is an apparently log-periodic oscillation below the expected depth, which can be seen in the observation for 5000 buckets in Figure 16, as well as in Figures 21 and 20.

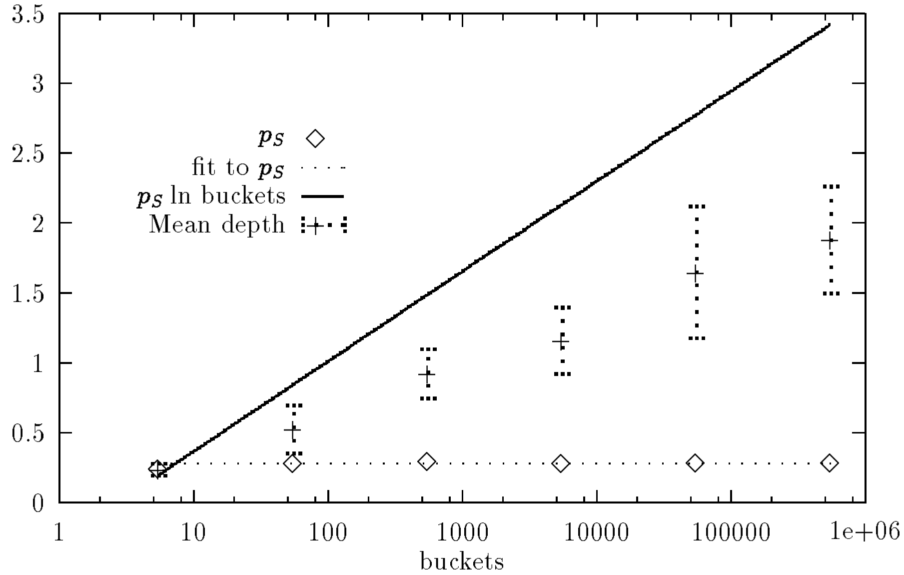

Figure 19: Simulation with uniform data distribution. Bucket size = 5

7 Conclusion and further work

The ***Capitalist model*** is a useful technique for modeling certain multi-dimensional and spatial file structures.

We have applied it to NIBGF and BANG files, to build models of file growth and storage utilization.

Our model predicts that the expected height of a node in the directory tree is $p_S \ln(\text{file size})$ where $p_S \leq 1$ is function of the bucket size. Based on the analytic models, we deprecate the importance of binary search between containment levels in the tree. Within a single level, the value of binary search is confirmed, since a directory node may have thousands of children.

Our models do not explain a number of second-order features of the algorithms' behavior which appear in simulations. These remain to be examined. Future work on the ***Capitalist*** model will include models of other spatial data structures.

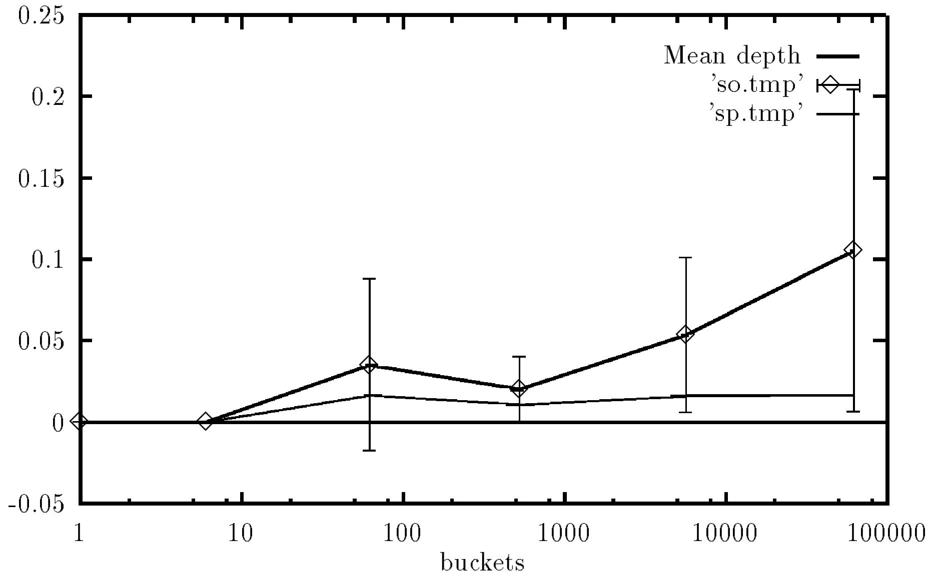

Figure 20: Simulation with uniform distribution. Bucketsize = 50

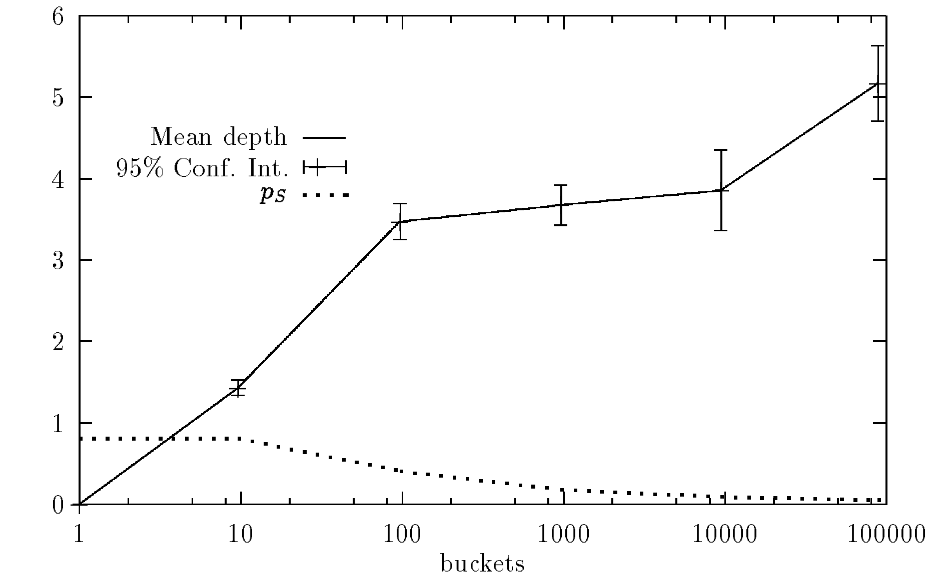

Figure 21: Simulation with normal distribution. Bucketsize = 30

References

[AS64] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions*. Dover, 1964.

[Ben75] J.L. Bentley. Multidimensional binary search trees used for associative searching. *CACM*, 18(9), September 1975.

[BY89] Ricardo A. Baeza-Yates. Modeling splits in files structures. *Acta Informatica*, 26(4):349–362, February 1989.

[Com79] Douglas Comer. The ubiquitous B-tree. *Computing Surveys*, 11(2), June 1979.

[FB74] R.A. Finkel and J.L. Bentley. Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

[Fla83] Philippe Flajolet. On the performance evaluation of extendible hashing and trie searching. *Acta Informatica*, 20(4), 1983.

[FNPS79] Ronald Fagin, Jurg Nievergelt, Nicholas Pippenger, and H. Raymond Strong. Extendible hashing – a fast access method for dynamic files. *ACM Transactions on Database Systems*, 4(3), 1979.

[Fre87] Michael Freeston. The BANG file: a new kind of grid file. In *1987 ACM-SIGMOD Conference*, pages 260–269, 1987.

[GKP88] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics / A Foundation for Computer Science*. Addison-Wesley, 1988.

[HB92] Nabil I. Hachem and P. Bruce Berra. New order preserving access methods for very large files derived from linear hashing. *IEEE Transactions on Knowledge and Data Engineering*, 4(1), February 1992.

[Ive62] Kenneth E. Iverson. *A Programming Language*. Wiley, 1962.

[Knu73] Donald E. Knuth. *The art of computer programming, Vol 3, Sorting and Searching*. Addison-Wesley, 1973.

[Lar85] Per-Åke Larson. Performance analysis of a single-file version of linear hashing. *Computer Journal*, 28(3), 1985.

[Lit80] Witold Litwin. Linear hashing: a new tool for file and table addressing. In *VLDB*, pages 212–223, 1980.

[Lom81] David B. Lomet. Digital B-trees. In *Proceedings of the 7th Conference on Very Large Data Bases*, pages 333–343, September 1981.

[LS90] David B. Lomet and Betty Salzberg. The hB-tree: A multiattribute indexing method with good guaranteed performance. *TODS*, 15(4), December 1990.

[NHS84] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1), March 1984.

[OM91] Aris Ouksel and Otto Mayer. The nested interpolation based grid file. In *Mathematical Fundamentals of Database Systems*, pages 173–187. Springer-Verlag, 1991.

[OM92] M. Aris Ouksel and Otto Mayer. A robust and efficient spatial data structure. *Acta Informatica*, 29, 1992.

[Ore83] J. Orenstein. Multidimensional tries used for associative searching. In *Proceecdings of the ninth International Conference on Very Large Data Bases*, pages 132–141, 1983.

[Reg85] Mireille Regnier. Analysis of grid file algorithms. *BIT*, 25:335–357, 1985.

[Tay] Stephen Taylor. An approach for analysis of multidimensional files. Dissertation proposal and on-going work.

[Yao79] Andrew Chi-Chih Yao. On random 2-3 trees. *Acta Informatica*, 9(2):159–170, 1979.