10 Distributed Data Management I

There are two major approaches for the management of data in distributed systems: hashing and caching. The *hashing* approach tries to minimize the use of communication hardware by distributing data randomly among the given processors, and the *caching* approach tries to minimize the use of communication hardware by keeping data as close to the requesting processors as possible. In this section, we will concentrate on the hashing approach. First, we introduce a suitable class of hash functions.

10.1 Universal hashing

We need a class of functions that behave like random functions. The following definition formalizes this wish. ($[m]$ denotes the set $\{0,\ldots,m-1\}$.)

Definition 10.1 *Let $\mathcal{H} \subseteq \mathcal{F} = \{f :\ f : [p] \to [n]\}$ and let $k \in \mathbb{N}$. $\mathcal{H}$ is called a (α,k)-*universal *class of hash functions if, for all $\ell \le k$, all $\{x_1,\ldots,x_\ell\} \subseteq [p]$, and all $j_1,\ldots,j_\ell \in [n]$ we have*

$$\Pr[h(x_i) = j_i \text{ for all } i \in \{1,\ldots,\ell\}] \le \left(\frac{\alpha}{n}\right)^\ell .$$

*The probability is taken according to the random experiment of choosing a random function h out of $\mathcal{H}$. If $\alpha = 1$, we simply call $\mathcal{H}$ k-*universal*.*

Note that if h is a random function out of $\mathcal{F}$, then the probability above is equal to $(1/n)^\ell$. Next we present a class of universal hash functions. In the following, we assume p to be a prime number.

Definition 10.2 *Let $d \in \mathbb{N}$. For $a = (a_d,\ldots,a_0) \in [p]^{d+1}$ let*

$$h_a(x) = \left(\left(\sum_{i=0}^{d} a_i x^i\right) \bmod p\right) \bmod n$$

for all $x \in [p]$. Let $\mathcal{H}_a^d = \{h_a : [p] \to [n] :\ a \in [p]^{d+1}\}$.

Thus, $\mathcal{H}_a^d$ consists of all polynomials of degree at most d in $\mathbb{Z}_p$. The following result can be shown for $\mathcal{H}_a^d$.

Theorem 10.3 *$\mathcal{H}_a^d$ is $(1+o(1), d+1)$-universal.*

The reason why $\mathcal{H}_a^d$ is not $(1,d+1)$-universal is due to rounding effects, since p/n may not be an integer. Since these effects are fairly small if p is sufficiently large compared to n, we will assume $\mathcal{H}_a^d$ to be simply $(d+1)$-universal in the following.

10.2 Data management via hashing

In order to distribute the data among the n processors of a parallel system, we use a hash function $h : [p] \to [n]$ that is chosen randomly out of $\mathcal{H}_a^d$. Then we obtain the following result.

Theorem 10.4 *We are given a network G with routing number R and a random hash function out of $\mathcal{H}_a^d$, $d = \Theta(\log n)$, to distribute the memory cells among the processors. Suppose that the processors want to access n pairwise different memory cells, one per processor. Then there are paths for this with congestion and dilation at most $O(R + \log n)$.*

Proof. We know that for G there exists a path system $\mathcal{P}$ (that has one path for every pair of nodes) with dilation at most R and congestion at most $n \cdot R$. Sending requests to n distinct memory cells corresponds to selecting n random paths out of $\mathcal{P}$, one path per source node. Obviously, the dilation of these paths is at most R. Hence, it remains to bound the congestion.

Consider an arbitrary edge e in G. For every path p crossing e, let the binary random variable X_p be 1 if and only if p has been selected. Let us group the paths into $k = R/\log n$ sets $P_1, \ldots, P_k$ so that each set contains at most $n \log n$ paths. For each path set P_i, let $C_i = \sum_{p \in P_i} X_p$. Since $\Pr[X_p = 1] = 1/n$, it holds that $\mathrm{E}[C_i] = |P_i|/n \le \log n$ for every i. We will show that with high probability the number of paths out of P_i crossing e will not be much larger than $\mathrm{E}[C_i]$ w.h.p. Because h is chosen from a $\Theta(\log n)$-universal class of hash functions, we get for $\ell \ge 2\mathrm{e} \log n$ that

$$
\begin{aligned}
&\Pr[\text{at least } \ell \text{ paths are chosen from } P_i] \\
&\quad = \Pr[\bigcup_{i_1 < \ldots < i_\ell} (\text{paths } p_{i_1}, \ldots, p_{i_\ell} \text{ are chosen from } P_i)] \\
&\quad \le \sum_{i_1 < \ldots < i_\ell} \Pr[\text{paths } p_{i_1}, \ldots, p_{i_\ell} \text{ are chosen from } P_i] \\
&\quad \overset{d \ge \ell}{\le} \binom{n \log n}{\ell} \left(\frac{1}{n}\right)^\ell \le \left(\frac{\mathrm{e} n \log n}{\ell}\right)^\ell \cdot \left(\frac{1}{n}\right)^\ell \\
&\quad \le \left(\frac{1}{2}\right)^\ell \le \frac{1}{n^4} .
\end{aligned}
$$

Hence, the probability that there is a path set P_i with at least $2\mathrm{e} \log n$ paths is at most $(R/\log n)/n^4 \le 1/n^3$. Thus, the probability that the congestion at e exceeds $2\mathrm{e}R$ is at most $1/n^3$, and therefore the probability that the congestion at any of the at most n^2 edges exceeds $2\mathrm{e}R$ is at most $1/n$. This completes the proof. □

If we had a good on-line routing algorithm for arbitrary graphs, Theorem 10.4 would imply a low running time.

10.3 Data replacement strategies

In a parallel system it may happen frequently that the number of processors available for the storage of the memory cells changes frequently. This may be due to the fact that several parallel programs are executed concurrently by the parallel system, and these programs have varying resource requirements, making it necessary to redistribute the resources (such as processors

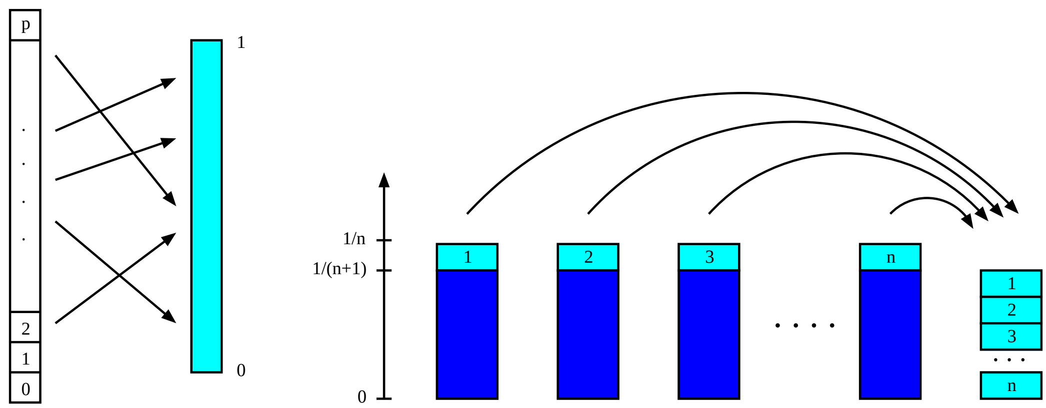

Figure 1: The uniform mapping scheme.

and memory) among them. Also, breakdowns of processors may occur, making it necessary to recover and replace data at the remaining processors.

Thus, a hashing strategy is needed that can adjust efficiently to a changing number of processors. Such a hashing strategy will be presented in the following.

The cut-and-paste strategy

The placement scheme consists of two stages and is based on a hash function $f : [p] \to [0, 1)$ chosen randomly out of some sufficiently good class of hash functions $\mathcal{H}$. (The use of the interval $[0, 1)$ instead of some interval $[n]$ simplifies the arguments below. It is easy to get from one representation to the other.) First, f is used to map the memory blocks to real numbers in the range $[0, 1)$. Since f is a random function, this guarantees that for any subset of $[0, 1]$ of range s, the relative amount of active memory blocks that are assigned to a number in this subset if equal to s. Thus, it only has to be ensured that the range $[0, 1)$ is distributed among the storage devices in such a way that every storage device gets a part of this range of size $1/n$ (see Figure 1).

For the mapping of the $[0, 1)$ range to the storage devices, a so-called *cut-and-paste function* is used. This function will take care that every processor has the same share of the $[0, 1]$ interval. To simplify the description, given n disks, we will denote the set of ranges assigned to disk i simply by $[0, 1/n]_i$. In the case of a step-wise increase in the number of storage devices from 1 to N, the cut-and-paste function works as follows:

At the beginning, we assign the whole range $[0, 1)$ to storage device 1. The *height* of a memory cell c in this case is defined as $h(c)$. For the change from n to $n+1$ processors, $n \in \{1, \ldots, N-1\}$, we cut off the range $[1/(n+1), 1/n]_i$ from every processor i and concatenate these intervals to a range $[0, 1/(n+1)]_{n+1}$ for the new processor $n+1$. What this actually means for the movement of memory blocks is described in Figure 2.

If one processor is lost, say processor i, then we reverse the replacement scheme in a way that first processor n moves all its cells back to the processors 1 to $n-1$ and then takes over the role of processor i (i.e., it obtains the number i and stores all memory cells processor i is required to store). This ensures the following result.

Replacement from n to $n+1$ processors:

for every processor $i \in \{1, \ldots, n\}$:

  for all memory blocks b at i with height $h \geq 1/(n+1)$:

    move b to device $n+1$

    the new height of b is $h - \frac{1}{n+1} + \frac{n-i}{n(n+1)}$

Figure 2: The replacement scheme for a new processor.

Theorem 10.5 *Suppose that the expected fraction of memory cells that fall into any part of the $[0,1)$ interval of size s is equal to s. Then the expected fraction of memory cells moved by the cut-and-paste strategy is equal to $1/n$ when increasing the number of processors by one, and equal to at most $2/n$ when decreasing the number of processors by 1, where n is the new number of processors.*

Hence, the cut-and-paste strategy involves only the replacement of a small number of blocks. Furthermore, it guarantees the following invariant:

Invariant: In *any* situation in which we have n processors, the memory cells are distributed among them as if this would be if we had a step-wise extension from 1 to n processors.

In order to compute the actual position of a memory cell, it therefore suffices to replay the cut-and-paste scheme for a step-wise increase from 1 to n processors. Fortunately, we do not have to go through all steps for a memory cell, but only have to consider those steps that require the cell to be replaced. In this case, we obtain the algorithm given in Figure 3 to compute the position of a cell.

Input: cell number $c \in [p]$
Output: number of processor $d \in \{1, \ldots, n\}$

Algorithm:

set $d = 1$ and $x = f(c)$

while $x \geq 1/n$ do

  set $y = \lceil 1/x \rceil$

  set $x = x - 1/y + (y-1-d)/(y(y-1))$

  set $d = y$

Figure 3: The computation of the position of a cell.

It is not difficult to show that for any number d of a processor in which a cell is currently stored, the second next processor at which it will be stored is at least $2d$. Hence, the number of rounds needed for the computation of the position of a memory cell is $O(\log n)$. Hence, we found a hashing method that can handle efficiently a changing number of processors.

10.4 References

- J. Carter and M. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.

- A. Brinkmann, K. Salzwedel, and C. Scheideler. Efficient distributed data placement strategies for storage area networks. In *Proc. of the 12th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2000.

- F. Meyer auf der Heide, C. Scheideler, and V. Stemann. Exploiting storage redundancy to speed up randomized shared memory simulations. *Theoretical Computer Science* 162, July 1996.

- A. Siegel. On universal classes of high performance hash functions, their time-space trade-off, and their applications. In *Proc. of the 30th IEEE Conference on Foundations of Computer Science (FOCS)*, pp. 20–25, 1989.