- **Update**: Update a record by replacing the value of one field.
- **Read**: Read a record, either one randomly chosen field or all fields.
- **Scan**: Scan records in order, starting at a randomly chosen record key. The number of records to scan is randomly chosen.

For scan specifically, the distribution of scan lengths is chosen as part of the workload. Thus, the `scan()` method takes an initial key and the number of records to scan. Of course, a real application may instead specify a scan interval (i.e., from February 1st to February 15th). The number of records parameter allows us to control the size of these intervals, without having to determine and specify meaningful endpoints for the scan. (All of the database calls, including `scan()`, are described in Section 5.2.1.)

4.1 Distributions

The workload client must make many random choices when generating load: which operation to perform (Insert, Update, Read or Scan), which record to read or write, how many records to scan, and so on. These decisions are governed by random distributions. YCSB has several built-in distributions:

- **Uniform**: Choose an item uniformly at random. For example, when choosing a record, all records in the database are equally likely to be chosen.
- **Zipfian**: Choose an item according to the Zipfian distribution. For example, when choosing a record, some records will be extremely popular (the *head* of the distribution) while most records will be unpopular (the *tail*).
- **Latest**: Like the Zipfian distribution, except that the most recently inserted records are in the head of the distribution.
- **Multinomial**: Probabilities for each item can be specified. For example, we might assign a probability of 0.95 to the Read operation, a probability of 0.05 to the Update operation, and a probability of 0 to Scan and Insert. The result would be a read-heavy workload.

Figure 1 illustrates the difference between the uniform, zipfian and latest distributions. The horizontal axes in the figure represent the items that may be chosen (e.g., records) in order of insertion, while the vertical bars represent the probability that the item is chosen. Note that the last inserted item may not be inserted at the end of the key space. For example, Twitter status updates might be clustered by user, rather than by timestamp, meaning that two recently inserted items may be far apart in the key space.

A key difference between the Latest and Zipfian distributions is their behavior when new items are inserted. Under the Latest distribution, the newly inserted item becomes the most popular, while the previously popular items become less so. Under the Zipfian distribution, items retain their popularity even as new items are inserted, whether or not the newly inserted item is popular. The Latest distribution is meant to model applications where recency matters; for example, only recent blog posts or news stories are popular, and the popularity decays quickly. In contrast, the Zipfian distribution models items whose popularity is independent of their newness; a particular user might be extremely popular, with many views of her profile page, even though she has joined many years ago.

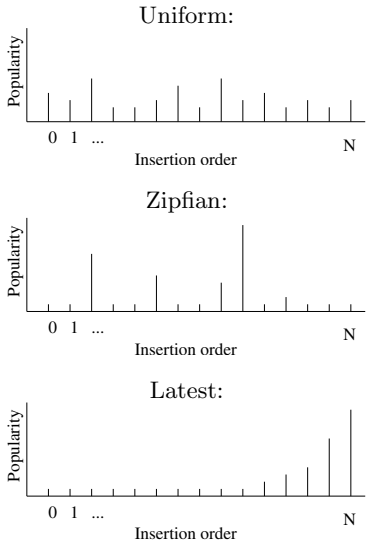

Figure 1: Probability distributions. Horizontal axes represents items in order of insertion, and vertical axes represent probability of being chosen.

4.2 The Workloads

We defined the workloads in the core package by assigning different distributions to the two main choices we must make: which operation to perform, and which record to read or write. The various combinations are shown in Table 2. Although we do not attempt to model complex applications precisely (as discussed above), we list a sample application that generally has the characteristics of the workload.

Loading the database is likely to take longer than any individual experiment. In our tests, loads took between 10-20 hours (depending on the database system), while we ran each experiment (e.g., a particular workload at a particular target throughput against a particular database) for 30 minutes. All the core package workloads use the same dataset, so it is possible to load the database once and then run all the workloads. However, workloads A and B modify records, and D and E insert records. If database writes are likely to impact the operation of other workloads (e.g., by fragmenting the on-disk representation) it may be necessary to re-load the database. We do not prescribe a particular database loading strategy in our benchmark, since different database systems have different loading mechanisms (including some that have no special bulk load facility at all).

5. DETAILS OF THE BENCHMARK TOOL

We have developed a tool, called the *YCSB Client*, to execute the YCSB benchmarks. A key design goal of our tool is extensibility, so that it can be used to benchmark new cloud database systems, and so that new workloads can be developed. We have used this tool to measure the performance of several cloud systems, as we report in the next section. This tool is also available under an open source license, so that others may use and extend the tool, and contribute new workloads and database interfaces.