- Yahoo! Cloud Serving Benchmark (YCSB) framework, with the goal of facilitating **performance comparisons** of the new generation **of cloud data serving systems.** We d**efine a core set of benchmarks** and **report results** for four widely used systems: Cassandra(NoSQL), HBase(NoSQL), Yahoo!'s PNUTS, and a simple sharded MySQL implementation
- NoSQL systems share the goals of massive scaling "on demand" (elasticity) and simplified application development and deployment.
- The most obvious **differences between the available data models** are between the various data models, such as the column-group oriented BigTable model used in Cassandra and HBase versus the simple hashtable model of Voldemort or the document model of CouchDB.
- Data models can be compared easily, data systems on the other hand, cannot be done easily. Some systems have optimized writes while others for reads
- Developers of various systems report performance numbers for the "sweet spot" workloads for their system, which may not match the workload of a target application. So to verify, developers have to manually download and check each system for their application and choose the better performing one.
- YCSB creates a benchmark and benchmarking framework to compare cloud systems
- It focuses on serving systems rather than analytical systems(offline queries)
- There are existing benchmarks for a variety of data storage systems (such as SQL databases [22] and filesystems [12, 10]). However, the novel interfaces (usually neither SQL nor POSIX), elasticity, and new use cases of cloud serving systems motivate a new benchmark.

YCSB: The framework consists of a **workload generating client** and a **package of standard workloads** that cover interesting parts of the performance space (read-heavy workloads, write-heavy workloads, scan workloads, etc.).

- **Benchmark tiers**:
    1. **Performance**: focuses on the latency of requests when the database is under load.

        ->there is an inherent tradeoff between latency and throughput
        ->as the load increases, the latency of individual requests increases
        ->Typically application designers must decide on an acceptable latency, and provision enough servers to achieve the desired throughput while preserving acceptable latency.
        ->A system with better performance will achieve the desired latency and throughput with fewer servers.
        ->The Performance tier of the benchmark aims to characterize this tradeoff for each database system by measuring latency as we

increase throughput, until the point at which the database system is saturated and throughput stops increasing
->To conduct this benchmark tier, we need a **workload generator** which serves two purposes: first, to define the dataset and load it into the database; and second, to execute operations against the dataset while measuring performance. **YCSB Client** is implemented for both these actions.
-> It takes in throughput value as input so generating latency vs throughput graph becomes easy.

2. **Scaling:** A key aspect of cloud systems is their ability to scale elastically, so that they can handle more load as applications add features and grow in popularity. The Scaling tier of the database examines the impact on performance as more machines are added to the system. There are two metrics to measure in this tier:
    a. **Scaleup:** How does the database perform as the number of machines increases (the increase in the number of machines and data is done when no workload is running)
        ->the performance should remain constant, as the number of servers, amount of data, and offered throughput scale proportionally.
    b. **Elastic speedup:** How does the database perform as the number of machines increases while the system is running.
        -> The performance should increase as the new servers are being added.

- **Benchmark Workloads(**Present in the package):
    Some systems may be highly optimized for reads but not for writes, or for scans but not for point lookups, or, for inserts but not updates. The workloads in the core package were chosen to explore these tradeoffs directly
    ->Each operation against the data store is randomly chosen to be one of:
    • **Insert**: Insert a new record.
    • **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.

    1. **Workload A:** Update heavy workload: 50/50% Mix of Reads/Writes
    2. **Workload B:** Read mostly workload: 95/5% Mix of Reads/Writes
    3. **Workload C:** Read-only: 100% reads
    4. **Workload D:** Read the latest workload: More traffic on recent inserts
    5. **Workload E:** Short ranges: Short range based queries
    6. **Workload F:** Read-modify-write: Read, modify and update existing records