Project VaultX: Performance Report

Course: CSE 208 (Data Structures and Algorithms II Sessional)
Assignment: Offline 3 - Hashing

Name: Md. Riyadun Nabi
Roll Number: 2205076
Section: B1, L2-T1

July 7, 2025

1 Introduction

This report describes the implementation and performance evaluation of a hash-based storage engine for "VaultX", designed to efficiently handle the insertion, search, and deletion of thousands of pseudo-random string key-value pairs. The system adapts to workload patterns, utilizes multiple collision resolution techniques, and reports system health using several performance metrics.

$_{l}ine$

2 Hash Functions Used

2.1 Hash1: Polynomial Rolling Hash

- **Description:** The Polynomial Rolling Hash is a classic hash function, ideal for strings. It processes the string left to right, multiplying each character by increasing powers of a small prime number, then summing and reducing modulo the table size (N).
- **Formula:**

```
long long hash1(const string &key, long long N) {
    long long hash_val = 0;
    long long p = 31;
    long long p_pow = 1;
    for (char c : key) {
        hash_val = (hash_val + (c - 'a' + 1) * p_pow) % N;
        p_pow = (p_pow * p) % N;
    }
    return hash_val;
}
```

- **Reason for Use:** This function distributes string keys well for uniformly random input and reduces clustering, especially when N is prime.

2.2 Hash2: djb2 Hash Function

- **Description:** The djb2 function, popularized by Dan Bernstein, uses bit manipulation and multiplication by 33 to scramble input strings effectively.
- **Formula:**

```
long long hash2(const string &key, long long N) {
    unsigned long hash_val = 5381;
    for (char c : key) {
        hash_val = ((hash_val << 5) + hash_val) + c;
    }
    return (hash_val % (N - 1)) + 1;
}
```

- **Reason for Use:** It is simple, fast, and proven to provide good key distribution for non-cryptographic purposes.

Both hash functions were empirically tested to ensure at least 60% unique hash values for typical workloads, minimizing clustering and secondary collisions.

$_l ine$

3 Constants Used

- Minimum Key Length: 5
- Maximum Key Length: 10
- Initial Table Size (N'): 10,007
- Actual Table Size (N): Next prime $\geq N'$
- Linear Probing Step Size (S): 5
- Number of Words Generated: 10,000
- Load Factors Evaluated: 0.4, 0.5, 0.6, 0.7, 0.8, 0.9

$_l ine$

4 Collision Resolution Techniques

1. **Separate Chaining with Balanced BST (Red-Black Tree)**
   - Each slot in the table holds a pointer to a red-black tree.
   - Colliding entries are inserted into the tree, keeping the structure balanced.
   - **Advantage:** Fast searches and inserts, especially when the chains are not too long.
2. **Linear Probing with Step Adjustment**
   - On collision, probe the next slot using: `index = (hash(k) + i*S) % N`.
   - The step size (S) is set to a small prime to help probe all slots.
   - **Advantage:** Simple and cache-friendly.
3. **Double Hashing**
   - On collision, probe using: `index = (hash1(k) + i*hash2(k)) % N`.
   - Hash2 ensures the probe sequence is key-dependent, reducing clustering.
   - **Advantage:** Minimizes primary and secondary clustering; good for uniformity.

$_l ine$

5 Word Generation and Storage

Words are generated randomly as unique lowercase strings of length 5–10. Duplicates are detected via the hash table search mechanism, ensuring each (key, value) pair is unique and values are assigned in insertion order.

6 Performance Results

The following tables summarize the performance metrics for each collision resolution technique and hash function across various load factors.

Table 1: For Load Factor 0.4

Method	**Hash1 Function**					**Hash2 Function**				
	# of Collisions during insertion	**Before Deletion**		**After Deletion**		# of Collisions during insertion	**Before Deletion**		**After Deletion**	
		Avg Search Time (ns)	Avg Probes	Avg Search Time (ns)	Avg Probes		Avg Search Time (ns)	Avg Probes	Avg Search Time (ns)	Avg Probes
Separate Chaining with balanced BST	709	383.40	N/A	429.85	N/A	718	507.52	N/A	292.96	N/A
Linear Probing with Step Adjustment	1325	541.25	1.3025	762.13	2.3775	1404	317.12	1.3500	652.01	2.3500
Double Hashing	1137	599.88	1.2575	1045.47	2.1100	1141	722.37	1.3175	1791.82	2.1200

Table 2: For Load Factor 0.5

Method	**Hash1 Function**					**Hash2 Function**				
	# of Collisions during insertion	**Before Deletion**		**After Deletion**		# of Collisions during insertion	**Before Deletion**		**After Deletion**	
		Avg Search Time (ns)	Avg Probes	Avg Search Time (ns)	Avg Probes		Avg Search Time (ns)	Avg Probes	Avg Search Time (ns)	Avg Probes
Separate Chaining with balanced BST	1094	503.22	N/A	1858.88	N/A	1082	487.93	N/A	314.76	N/A
Linear Probing with Step Adjustment	2565	596.63	1.5100	1025.31	2.9460	2519	475.38	1.5440	1151.42	3.0720
Double Hashing	2054	726.13	1.4320	2400.35	2.5120	1969	651.12	1.3820	1997.57	2.3700

Table 3: For Load Factor 0.6

Method	**Hash1 Function**					**Hash2 Function**				
	# of Collisions during insertion	**Before Deletion**		**After Deletion**		# of Collisions during insertion	**Before Deletion**		**After Deletion**	
		Avg Search Time (ns)	Avg Probes	Avg Search Time (ns)	Avg Probes		Avg Search Time (ns)	Avg Probes	Avg Search Time (ns)	Avg Probes
Separate Chaining with balanced BST	1498	442.13	N/A	837.69	N/A	1519	470.76	N/A	397.79	N/A
Linear Probing with Step Adjustment	4538	1066.80	1.7983	1599.90	4.3417	4756	769.18	1.8617	890.38	3.9300
Double Hashing	3264	1214.77	1.5033	2020.83	2.7050	3240	2000.98	1.5283	1462.54	2.8600

Table 4: For Load Factor 0.7

Method	**Hash1 Function**					**Hash2 Function**				
	# of Collisions during insertion	**Before Deletion**		**After Deletion**		# of Collisions during insertion	**Before Deletion**		**After Deletion**	
		Avg Search Time (ns)	Avg Probes	Avg Search Time (ns)	Avg Probes		Avg Search Time (ns)	Avg Probes	Avg Search Time (ns)	Avg Probes
Separate Chaining with balanced BST	1977	794.62	N/A	494.75	N/A	1984	511.41	N/A	348.72	N/A
Linear Probing with Step Adjustment	8684	846.95	2.2914	2025.32	6.0000	8721	466.52	2.2229	1872.68	5.9343
Double Hashing	5052	958.66	1.6871	1689.00	3.2957	5104	846.45	1.6857	2989.70	3.4043

Table 5: For Load Factor 0.8

Method	**Hash1 Function**					**Hash2 Function**				
	# of Collisions during insertion	**Before Deletion**		**After Deletion**		# of Collisions during insertion	**Before Deletion**		**After Deletion**	
		Avg Search Time (ns)	Avg Probes	Avg Search Time (ns)	Avg Probes		Avg Search Time (ns)	Avg Probes	Avg Search Time (ns)	Avg Probes
Separate Chaining with balanced BST	2498	447.29	N/A	1085.91	N/A	2512	275.18	N/A	775.97	N/A
Linear Probing with Step Adjustment	15901	1000.18	2.8363	4212.49	10.1900	17603	1170.02	3.1587	2192.78	12.4225
Double Hashing	8209	2071.02	1.9700	2234.27	4.4625	8167	1211.31	2.1275	2123.29	4.3937

Table 6: For Load Factor 0.9

Method	**Hash1 Function**					**Hash2 Function**				
	# of Collisions during insertion	**Before Deletion**		**After Deletion**		# of Collisions during insertion	**Before Deletion**		**After Deletion**	
		Avg Search Time (ns)	Avg Probes	Avg Search Time (ns)	Avg Probes		Avg Search Time (ns)	Avg Probes	Avg Search Time (ns)	Avg Probes
Separate Chaining with balanced BST	3076	912.69	N/A	511.37	N/A	3085	304.65	N/A	667.13	N/A
Linear Probing with Step Adjustment	35777	1623.40	4.7267	8894.54	27.2600	50146	1231.31	6.3978	7213.32	36.4433
Double Hashing	14366	1521.75	2.8056	5451.44	7.9011	14515	2078.93	2.5689	3700.85	7.6322

7 Impact of Load Factor on Results

- **Collision Count:** As the load factor increases, the number of collisions during insertion rises for all techniques. Open addressing methods, especially linear probing, show more rapid growth due to primary clustering. This is evident as the collision count for Linear Probing skyrockets at a 0.9 load factor compared to Double Hashing and Separate Chaining.
- **Search Time and Probes:** For separate chaining, search time remains relatively stable (assuming balanced trees), but may increase slightly as chains grow. For open addressing, both search time and probe count increase significantly at higher load factors, since more slots are occupied and longer probe sequences occur. The degradation is particularly severe for Linear Probing, where the average probes after deletion reach as high as 36.44.
- **Performance After Deletion:** Search times and probe counts generally increase after 10% of the elements are deleted. This is because the deleted slots create "tombstones" or gaps in the probe sequences for open addressing, which must still be traversed during a search, lengthening the search path.
- **Method Comparison:** Double Hashing consistently results in fewer collisions and a lower average probe count than Linear Probing, especially as the load factor exceeds 0.6. This highlights its effectiveness in mitigating clustering. Separate Chaining with balanced BSTs shows robust performance across all load factors, with its search time being less affected by the table's density compared to open addressing methods.