**High-Performance Batch Processing Framework Architecture**

**1. Overview**

This document describes the architecture of a high-performance batch processing framework that uses a master-worker model with dynamic workload distribution. The system efficiently processes large datasets by distributing work across multiple worker nodes while maintaining optimal resource utilization through multi-level parallelism.

**2. Architecture Components**

**2.1 Node Types**

* **Master Node**: Coordinates the overall workflow and task distribution
* **Worker Nodes**: Execute the assigned processing tasks
* **Query Node**: A specialized worker that performs initial data analysis and bucketing

**2.2 Key Concepts**

* **Hash Keys**: Unique identifiers assigned to each record
* **Buckets**: Logical groupings of hash keys for processing
* **Concurrency Setting**: Defines the total number of buckets
* **Worker Capacity (T)**: The number of buckets a worker can process in parallel

**3. Workflow**

**3.1 Data Preparation**

1. Each record in the dataset is assigned a unique hash key
2. Hash keys help ensure related records are processed together

**3.2 Query and Bucketing**

1. A designated query node examines the dataset
2. Based on the system's concurrency setting, the query node creates buckets of hash keys
3. The number of buckets equals the concurrency setting
4. Hash keys are distributed across buckets based on relevant criteria

**3.3 Capacity Reporting**

1. Each worker node reports its capacity value (T) to the master
2. T represents how many buckets the worker can process simultaneously
3. Capacity may vary between workers based on hardware capabilities

**3.4 Task Assignment**

1. The master node receives bucket information from the query node
2. The master analyzes worker capacity reports
3. Buckets are assigned to workers based on their reported capacity
4. If total worker capacity exceeds bucket count, some capacity remains unused
5. If bucket count exceeds total capacity, some buckets wait for processing

**3.5 Parallel Processing**

1. Workers process their assigned buckets according to their capacity
2. Each worker can process multiple buckets simultaneously (up to its T value)
3. Within each bucket, hash keys are processed sequentially

**3.6 Result Aggregation**

1. As workers complete bucket processing, results are returned to the master
2. The master aggregates results into a complete dataset
3. The master handles any necessary post-processing or storage

**4. Example Scenario**

**Given:**

* 5 buckets created (concurrency setting = 5)
* 3 worker nodes
* Each worker has capacity T=2

**Distribution:**

* Worker 1: Processes 2 buckets in parallel
* Worker 2: Processes 2 buckets in parallel
* Worker 3: Processes 1 bucket in parallel

**5. Benefits and Considerations**

**5.1 Benefits**

* **Efficient Resource Utilization**: Workers operate at their optimal capacity
* **Multi-level Parallelism**: Parallel processing between workers and within workers
* **Workload Balancing**: Dynamic allocation based on worker capabilities
* **Scalability**: Easy to add or remove workers to adjust overall system capacity
* **Fault Tolerance**: If a worker fails, the master can reassign its buckets

**5.2 Considerations**

* **Network Overhead**: Communication between master and workers
* **Hash Key Distribution**: Uneven distribution may lead to processing imbalances
* **Worker Capacity Variance**: Heterogeneous worker capabilities require careful load balancing
* **Failure Handling**: Need for mechanisms to detect and recover from worker failures

**6. Monitoring and Optimization**

* Track worker utilization and processing times
* Adjust concurrency settings based on system performance
* Optimize hash key distribution to balance bucket workloads
* Consider workload characteristics when assigning buckets to workers