**High-Performance Batch Processing Framework**

**Architecture, Workflow, and Failover Design**

**1. Executive Summary**

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. It includes robust failure detection and recovery mechanisms to ensure processing reliability.

**2. Architecture Components**

**2.1 Node Types**

* **Master Node**: Coordinates the overall workflow and task distribution
* **Worker Nodes**: Execute the assigned processing tasks, with one worker node serving as the query node

**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
* **Query Role**: A temporary role assigned to one of the worker nodes to perform initial data analysis and bucketing

**3. Processing Workflow**

![Workflow Diagram]

**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. The master assigns the query role to one of the available worker nodes
2. This designated worker examines the dataset
3. Based on the system's concurrency setting, the worker creates buckets of hash keys
4. The number of buckets equals the concurrency setting
5. Hash keys are distributed across buckets based on relevant criteria
6. The worker returns bucket information to the master
7. After completing the query role, the worker returns to standard processing duties

**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 worker that performed the query role
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 (with 1 unit of unused capacity)

**5. Heartbeat Mechanism**

**5.1 Implementation**

* **Frequency**: Each worker sends a heartbeat signal to the master at regular intervals (typically 3-5 seconds)
* **Content**: Heartbeats include:
  + Worker ID
  + Current status (Active/Busy/Idle)
  + Health metrics (CPU usage, memory availability)
  + Currently processing bucket IDs
  + Completion percentage of each bucket
  + Any error conditions

**5.2 Monitoring**

* The master maintains a heartbeat registry with timestamps of the most recent heartbeat from each worker
* If a worker's heartbeat is not received within a configurable timeout period, the worker is considered potentially failed
* After missing multiple consecutive heartbeats, the worker is marked as failed

**6. Failover Architecture**

**6.1 Failure Detection**

* **Passive Detection**: Missing heartbeats from a worker
* **Active Detection**: Master periodically pings workers that haven't sent heartbeats
* **Confirmation**: Before declaring a worker failed, the master attempts to establish a direct connection

**6.2 State Preservation**

* Workers periodically checkpoint their progress for each bucket
* Checkpoints include:
  + Processed hash keys
  + Current processing state
  + Intermediate results
* Checkpoints are stored in a distributed storage system accessible by all nodes

**6.3 Task Recovery and Reassignment**

**6.3.1 Recovery Process**

1. When a worker is declared failed, the master identifies all affected buckets
2. For each affected bucket, the master determines the last successful checkpoint
3. The master updates its task registry to mark these buckets as requiring reassignment
4. Affected buckets are prioritized for reassignment

**6.3.2 Reassignment Strategies**

* **Immediate Reassignment**: Assigns failed buckets to workers with available capacity
* **Queue-Based Reassignment**: Places failed buckets in a priority queue for processing when capacity becomes available
* **Dynamic Capacity Adjustment**: Temporarily increases capacity (T value) of active workers

**6.4 Partial Bucket Recovery**

* If a bucket was partially processed, the new worker:
  1. Loads the checkpoint data
  2. Identifies already processed hash keys
  3. Continues processing only the remaining hash keys
  4. Merges new results with previously stored partial results

**6.5 Query Role Failure**

* If a worker fails while performing the query role:
  1. Master detects the failure through the heartbeat mechanism
  2. Master assigns the query role to another available worker
  3. The new worker restarts the query and bucketing process
  4. The system may use partial query results if available

**7. System Resilience Features**

**7.1 Master Node Redundancy**

* A standby master monitors the primary master via its own heartbeat mechanism
* Master state is replicated to the standby in near real-time
* Standby master can take over coordination if the primary fails

**7.2 Work Distribution Balancing**

* System periodically rebalances work across workers based on:
  + Processing speed
  + Current load
  + Health metrics

**7.3 Speculative Execution**

* For long-running buckets, the system may assign the same bucket to multiple workers
* The first worker to complete processing "wins"
* Reduces impact of stragglers and provides redundancy for critical tasks

**8. Benefits and Considerations**

**8.1 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
* **Fault Tolerance**: Robust recovery from worker failures
* **Scalability**: Easy to add or remove workers to adjust overall system capacity
* **Simplified Architecture**: No dedicated query node required, reducing system complexity

**8.2 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
* **Checkpoint Frequency Trade-off**: More frequent checkpoints improve recovery but increase overhead
* **Query Role Impact**: The worker performing query duties may have reduced processing capacity during this phase

**9. Monitoring and Optimization**

**9.1 Key Metrics**

* **Processing Throughput**: Records processed per second
* **Worker Utilization**: Percentage of worker capacity in use
* **Mean Time To Detect (MTTD)**: Average time to detect a worker failure
* **Mean Time To Recover (MTTR)**: Average time to reassign and resume processing of affected buckets
* **Recovery Success Rate**: Percentage of successfully recovered buckets

**9.2 Optimization Strategies**

* Adjust concurrency settings based on system performance
* Optimize hash key distribution to balance bucket workloads
* Tune checkpoint frequency based on failure patterns and processing characteristics
* Consider workload characteristics when assigning buckets to workers
* Select workers for the query role based on current load and processing capabilities