# High-Availability Algorithms for Distributed Stream Processing


Chenye Yang cy2540
Bingzhuo Wang bw2632
Zhuoyue Xing zx2269

# Introduction & System model

# Background and main focus

- In a distributed stream-processing systems(DSPS), the failure of a single server can significantly disrupt or even halt overall stream processing.

- We focus on approaches where once a server fails, a backup server takes over the operation of the failed one.

# The System Model

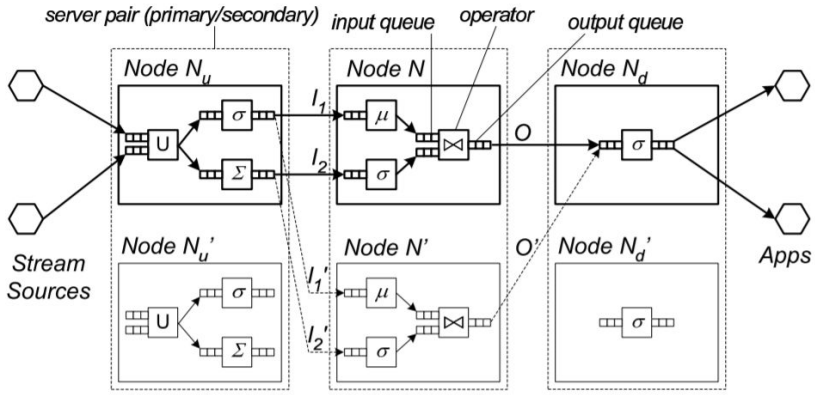



**Figure 1.** An example DSPS

- In stream-processing systems, each operator is a processing unit.
- A loop-free, directed graph of operators is called a query network.
- A DSPS partitions its query network across multiple nodes. Each node runs a stream-processing engine (SPE).

# The System Model

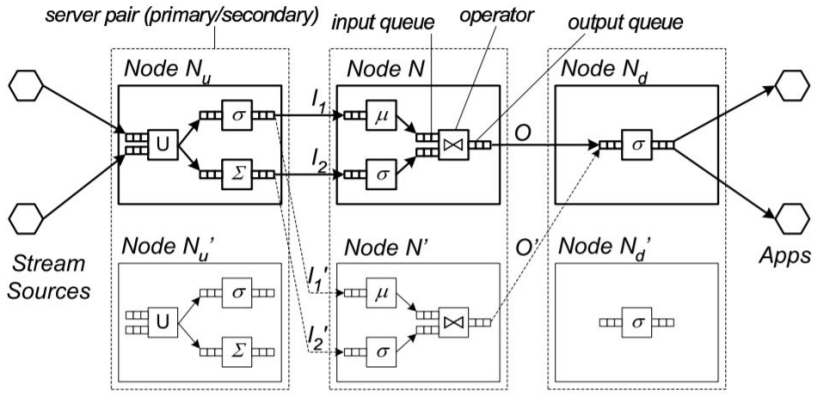



**Figure 1.** An example DSPS

- We focus on single-node fail-stop failures, we associate each node N with a recovery node N' that is in charge of detecting as well as handling the failure of N.
- Each recovery node runs its own SPE, and has the same query-network fragment as its primary, but its state is not necessarily the same as that of the primary.

# How to detect failures

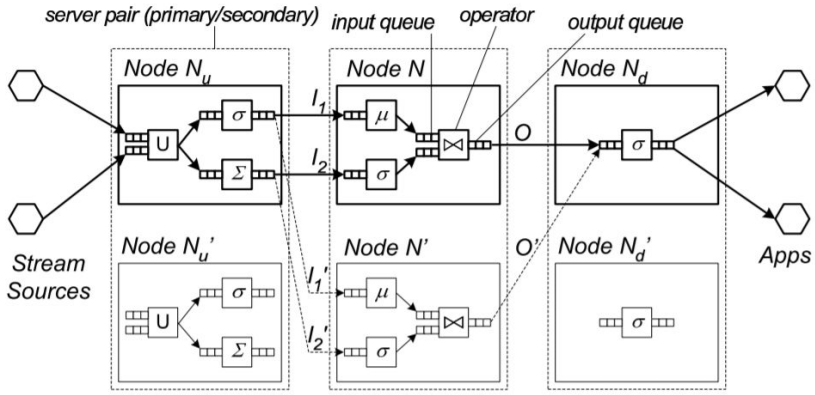



**Figure 1.** An example DSPS

- Each recovery node periodically sends keep-alive requests to its primary and assumes that the latter failed if a few consecutive responses do not return within a timeout period.
- When a recovery node detects the failure of its primary, if it was not already receiving the input streams, it asks the upstream nodes to start sending it the data.

# Methods

# Three types of recovery

Gap recovery
Allow information loss

Rollback recovery
Eliminate loss, allow duplicate

Precise recovery
Both duplicate and loss not allowed

## Four approaches to implement

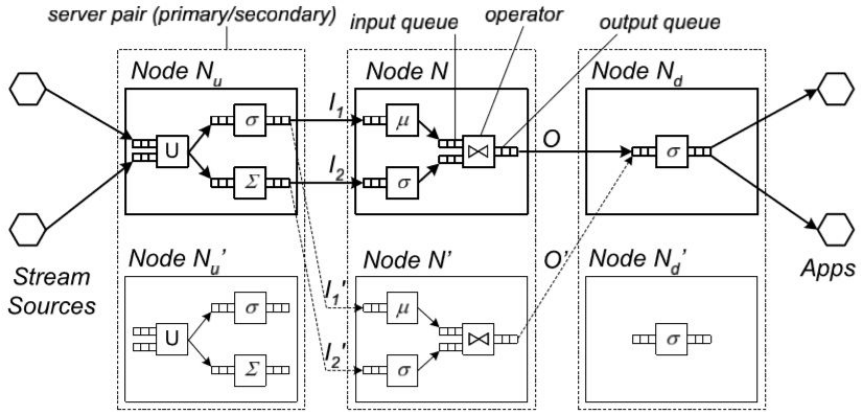
### Amnesia
Used in gap recovery

### Passive standby
Used in rollback and precise recovery

### Upstreaming backup
Used in rollback and precise recovery

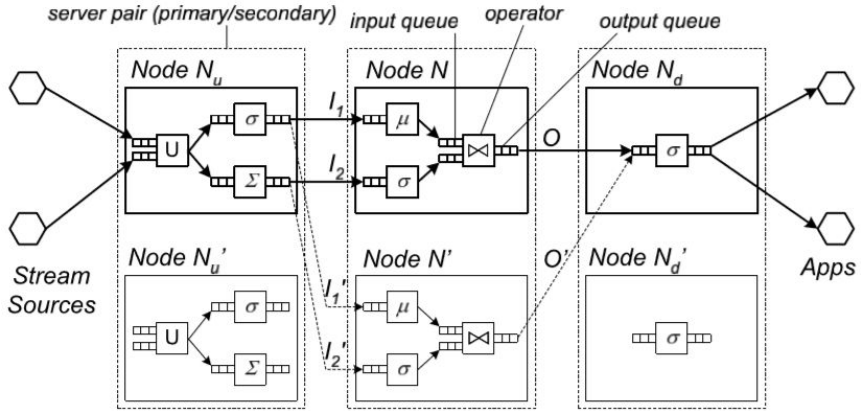
### Active standby
Used in rollback and precise recovery

# Gap recovery Amnesia



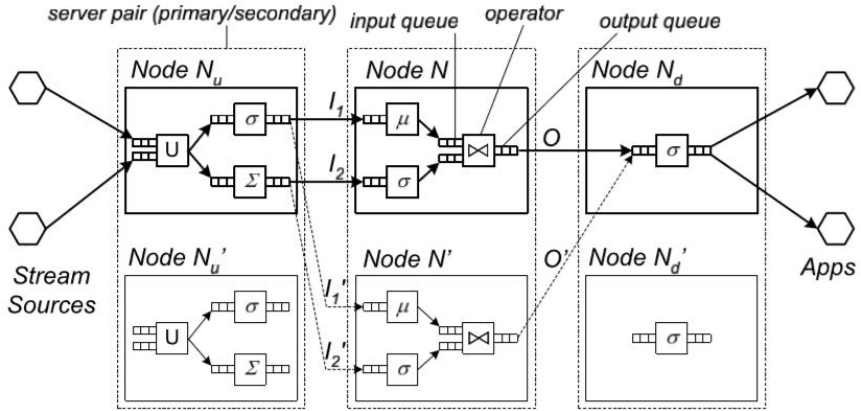

- Weakest constraint recovery
- Secondary node does not recreate the lost state and may drop tuples when the primary fails
- May result in information loss
- Zero recovery time
  The secondary reaches an equivalent state as soon as it discovers that the primary fails

# Rollback recovery Passive Standby



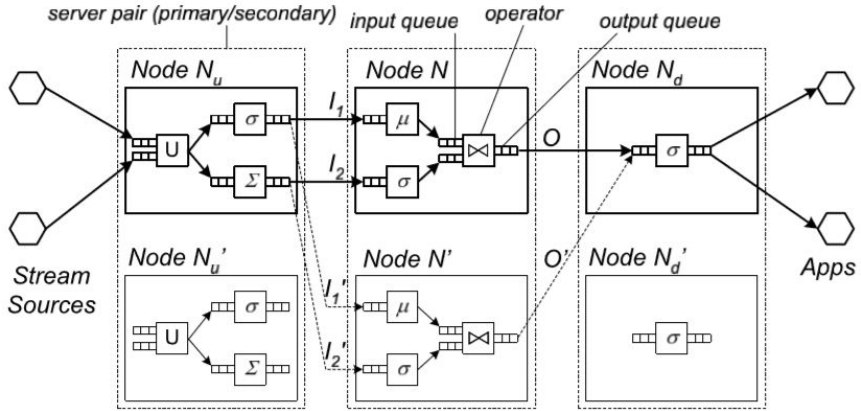

- Primary nodes periodically sends its latest checkpoint to the secondary
- Secondary nodes are always maintaining the last checkpointed state of primary
- If a primary fails, the secondary takes over--- duplicate occurs, no loss

# Rollback recovery Passive Standby



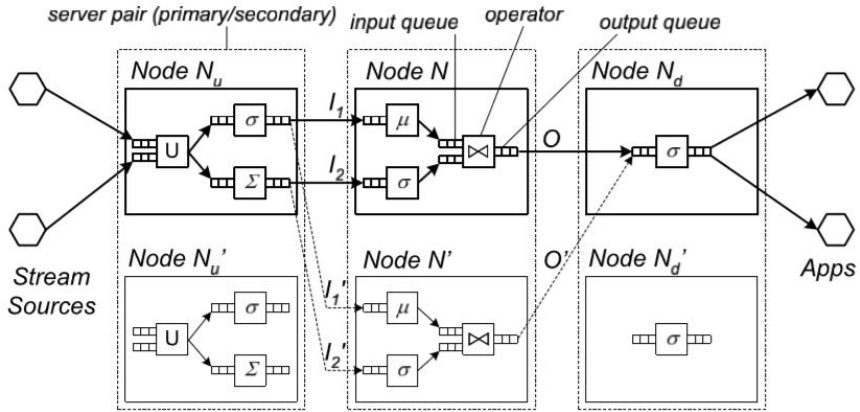

- Recovery time: K + Qp (delay time + duplicate re-processing time)
- K is the delay before the secondary node receives its first tuple
- Q is the number of duplicate input tuples it reprocesses
- P is the average processing time per input tuple
- Recovery time is relatively small

# Rollback recovery
# Upstreaming Backup



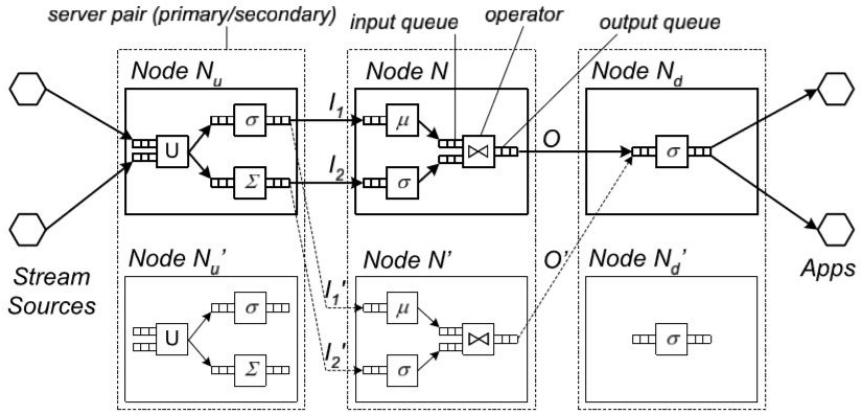

- Upstream nodes act as backups for their downstream neighbors
- Upstream nodes log all necessary tuples for the secondary to rebuild states
- If a primary fails, the secondary node reprocess the tuples logged at its upstream node

# Rollback recovery Upstreaming Backup



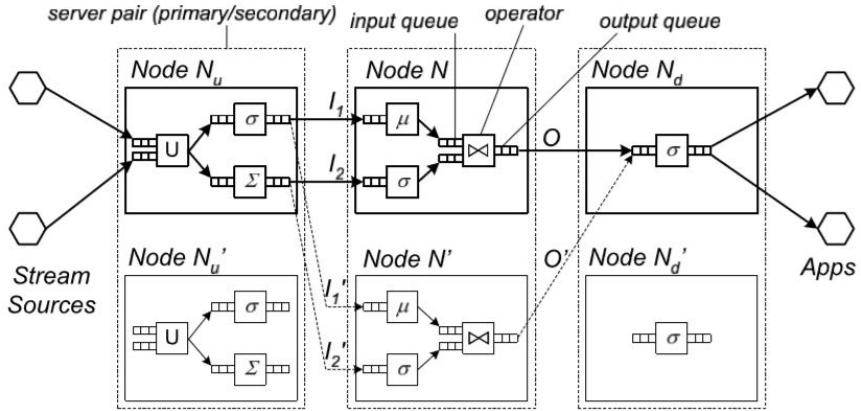

- Challenge: Determine the size of logged tuple that can be safely discarded
- Introduce inter-node communication to decide which part of the log can be discarded
- Recovery time has the same form as passive standby.
- While it needs to reprocess all tuples that contributed to the certain lost state, the de facto recovery time takes more

# Rollback recovery Active Standby



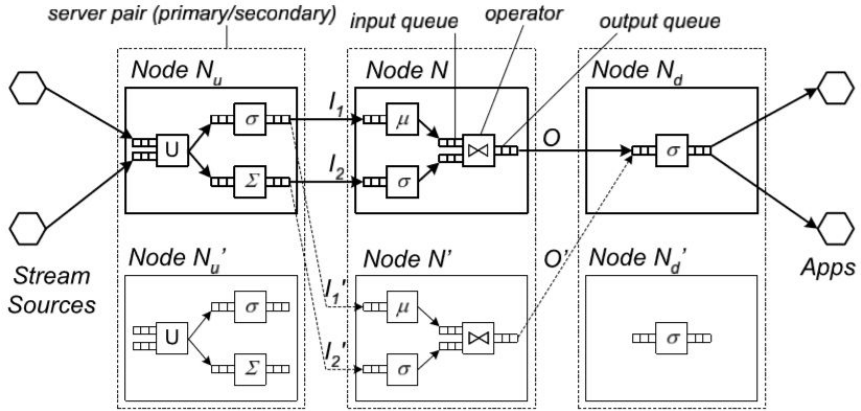

- The secondary node process in parallel with the primary from upstream nodes
- If a primary fails, the secondary node takes over and begins to send output to the downstream nodes
- Recovery time: negligible
  The recovery time lies in the transmission time of the output queue into downstreaming nodes

# Rollback recovery Active Standby



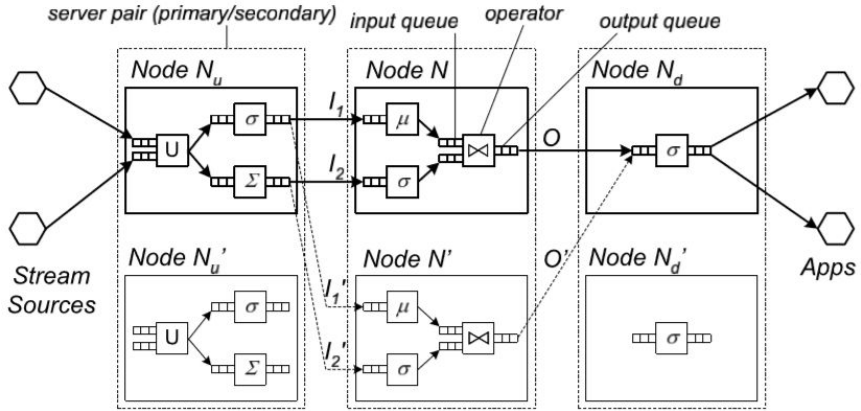

- Challenge: Identify the duplicate tuples between primary and secondary
- Introduce watermark, a second set of input tuple indicators, to define duplicate
- Watermark requires extra storage and processing time, which means the cost is relatively high

# Precise recovery
# Passive Standby



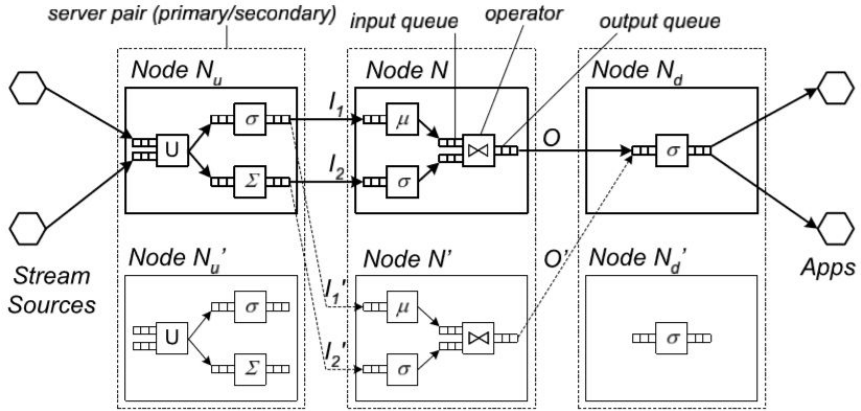

- Before sending output tuples, the failover node should ask all downstreaming nodes to discard tuples ahead of the checkpoint state
- May introduce more latency

# Precise recovery
# Upstream Backup




- Introduce watermarks to identify duplicate

# Precise recovery Active Standby




- Ensure that primary and secondary follow the same execution
- The primary should collect all information necessary to replay the execution
- Spend more time since the secondary must wait until the primary collect all information

# Evaluation

# Simulation Environment & Content

**Environment:**

Use CSIM18 to build a distributed stream-processing systems(DSPSs) for simple repeatable query networks.

Focus on Passive Standby, Active Standby, Upstream Backup. (Gap Recovery has no overhead and zero recovery time)

**Content:**

1. Runtime overhead and recovery time tradeoff
2. Cost of precise recovery
3. Effects of query-network properties on performance
4. Effects of query-network size on performance

# Runtime overhead and recovery time tradeoff

Overhead: represents the resources needed to finish the computation.

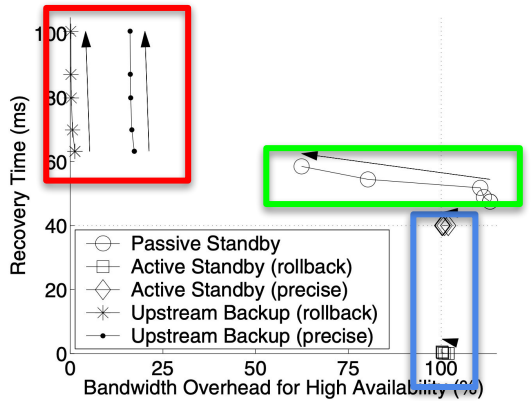



**Figure 6.** Recovery time and runtime overhead for rollback and precise recovery as the communication interval varies from 25 ms to 200 ms (indicated by the arrows)

1. Upstream Backup:
   a. Very little overhead (transmit only tuple identifier)
   b. Slowest recovery time (recreate the complete states of network)
   c. Recovery time sensitive to communication interval (More frequent the communication, more tuples are trimmed)
2. Active Standby:
   a. Overhead 100% or more (work simultaneously and transmitting identifier costs time)
   b. Negligible recovery time
3. Passive Standby:
   a. Medium recovery time (Secondary has checkpoint but needs reprocess some tuples and redirect the stream)
   b. Overhead sensitive to communication interval (Less frequent the communication, less checkpoints to transmit)

# Cost of precise recovery

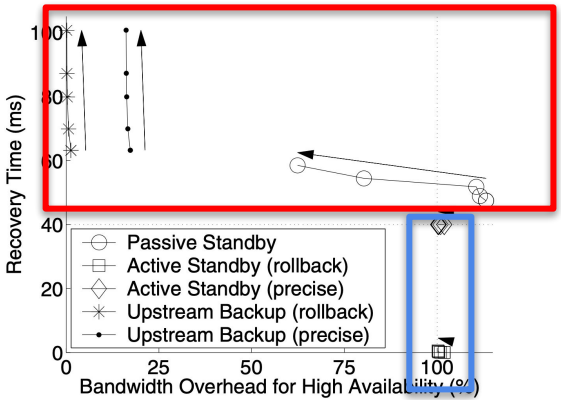



**Figure 6.** Recovery time and runtime overhead for rollback and precise recovery as the communication interval varies from 25 ms to 200 ms (indicated by the arrows)

Difference with Rollback Recovery Protocols:

Need to ask downstream for the latest tuples they received. Thus introduce delay/overhead.

For Upstream Backup and Passive Standby, communication happen in parallel with tuple reprocessing or stream redirection, thus add little delay.

For Active Standby, this delay cannot be masked, thus the recovery time extends by a constant value.

# Query-network properties vs performance

For rollback recovery, the query network type does not affect recovery time or runtime overhead.

| Query Network Type | Result | Upstream Backup | Active Standby | Passive Standby |
|---|---|---|---|---|
| Repeatable | Bw overhead (%) | 0.64 | 100.96 | 101.27 |
| | Rec. time (ms) | 47.62 | 1.80 | 45.88 |
| Convergent-capable | Bw overhead | 0.64 | 100.96 | **111.55** |
| | Rec. time | **69.86** | 0.07 | 48.88 |
| Non-deterministic | Bw overhead | 1.28 | 101.91 | 101.90 |
| | Rec. time | 50.92 | 1.82 | 47.24 |

**Table 6.** Effects of query-network type

# Query-network properties vs performance

| Window size (tuples) | 100 | 200 | 300 | 400 | 500 |
|---|---|---|---|---|---|
| PS overhead (%) | 111.55 | 111.55 | 111.54 | 111.54 | 111.54 |
| PS rec. time (ms) | 48.9 | 51.7 | 54.6 | 60.0 | 63.9 |
| UB rec. time (ms) | 69.9 | 98.9 | 138.7 | 188.5 | 248.3 |

**Table 7.** Effects of query-network state size

| Advance (tuples) | 100 | 50 | 25 | 10 | 5 |
|---|---|---|---|---|---|
| PS overhead (%) | 102.6 | 103.6 | 105.6 | 111.6 | 121.5 |
| PS rec. time (ms) | 47.5 | 47.5 | 47.6 | 48.8 | 51.6 |
| UB rec. time (ms) | 62.6 | 61.4 | 61.3 | 69.9 | 83.8 |

**Table 8.** Effects of rate of query-network state change

The properties that affect the performance:

1. Overhead:
   a. Upstream Backup and Active Standby: Rate of the message
   b. Passive Standby: Change of the checkpoint
2. Recovery time:
   a. Active Standby: Rate of the message
   b. Upstream Backup and Passive Standby: Processing complexity

# Query-network size vs performance

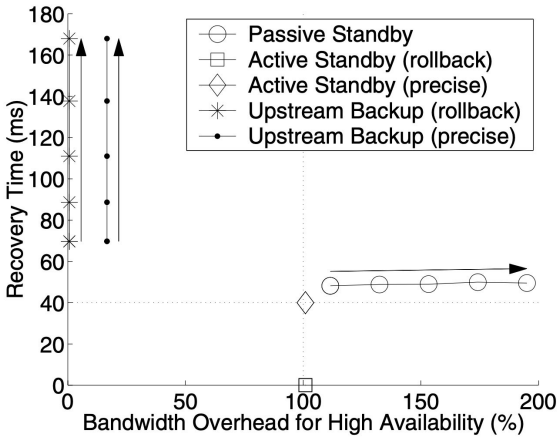



**Figure 7.** Effects of the number of operators. The arrows indicate the directions of the trends

Increasing the size and complexity of the query network translates into increasing the size of the query-network state, the rate at which this state changes, and the processing complexity.

Increasing the number of operators, which increases the state of network, will lead to the increase of recovery time of Upstream Standby (number of tuples increase) and Passive Standby (computation complexity increase), and will lead to the increase of overhead of Passive Standby (number of changed tuples in checkpoints increase)

# Discussion

**Active Standby**, provide the fastest recovery but at a high cost. Active Standby is thus best suited for environments where fast failure recovery justifies higher runtime costs, like **financial services, military applications**.

**Passive Standby** is best suited to provide precise recovery for arbitrary query networks, like **patient monitoring and other medical applications**, that impose a lower load on the system but necessitate precise recovery.

**Upstream Backup** has a significantly lower runtime overhead but a longer recovery time that depends mostly on the size of the query-network state. This approach is thus best suited for an environment where failures are infrequent and short recovery delays are tolerable, like **sensor-based environment and infrastructure monitoring applications**.