

SR3: Customizable Recovery for Stateful Stream Processing Systems

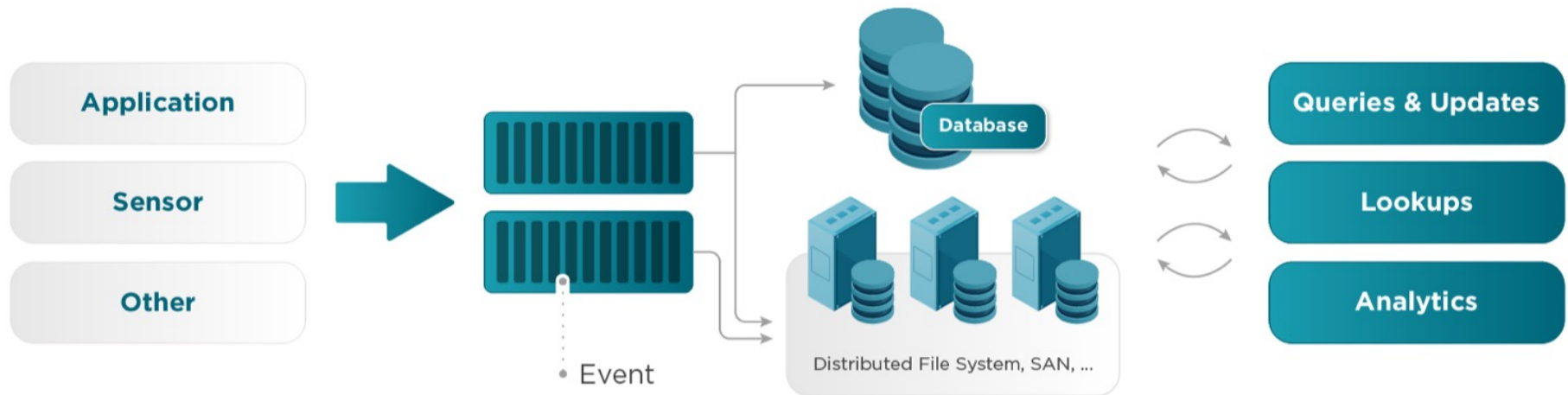
Hailu Xu, Pinchao Liu, Susana Cruz-Diaz
Dilma Da Silva‡, Liting Hu

California State University, Long Beach
Florida International University
Texas A&M University‡

Background – Stream Processing

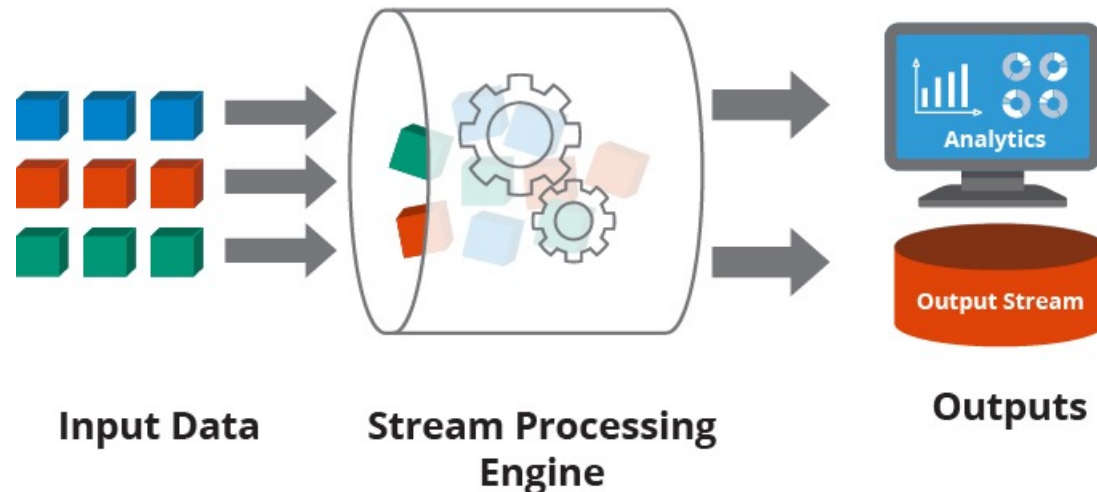
□ Before stream processing...

- ▶ Data was stored in a database, a file system, or other mass storage.
- ▶ Applications would query the data or compute over the data as needed.



Background

- Now we go to the era of stream processing...
 - ▶ applications and analytics react to events instantly
 - ▶ handle data volumes that are much larger
 - ▶ models the continuous and timely nature of most data



Background – Stream Processing System



Background

□ Let's look into the processing...

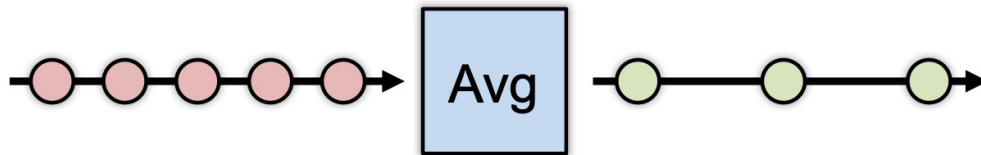
- ▶ Some applications or operations: **stateless**



- Function can filter inputs
 - if (input > threshold) { **emit** input }
- (1)

Background

□ Majority of streaming applications: **stateful**

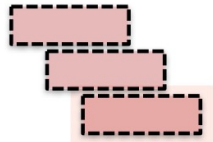


- E.g., Average value per window

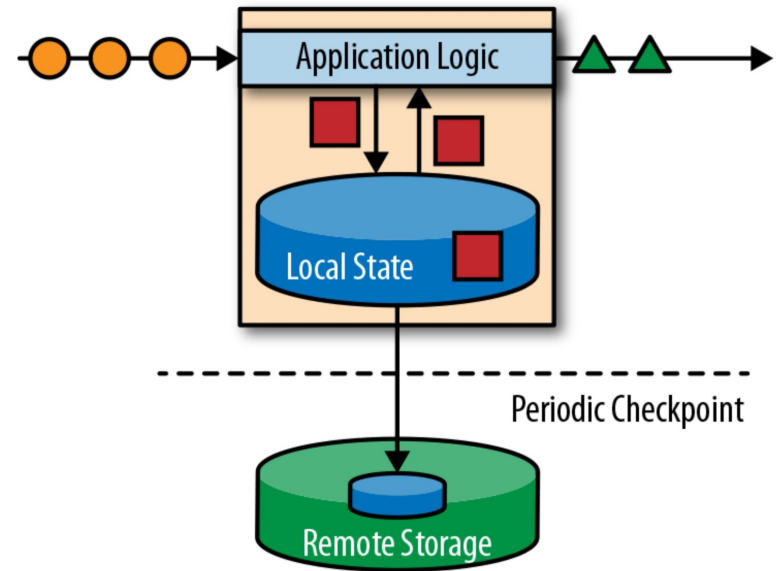
- Window can be # elements (10) or time (1s)

- Windows can be disjoint (every 5s)

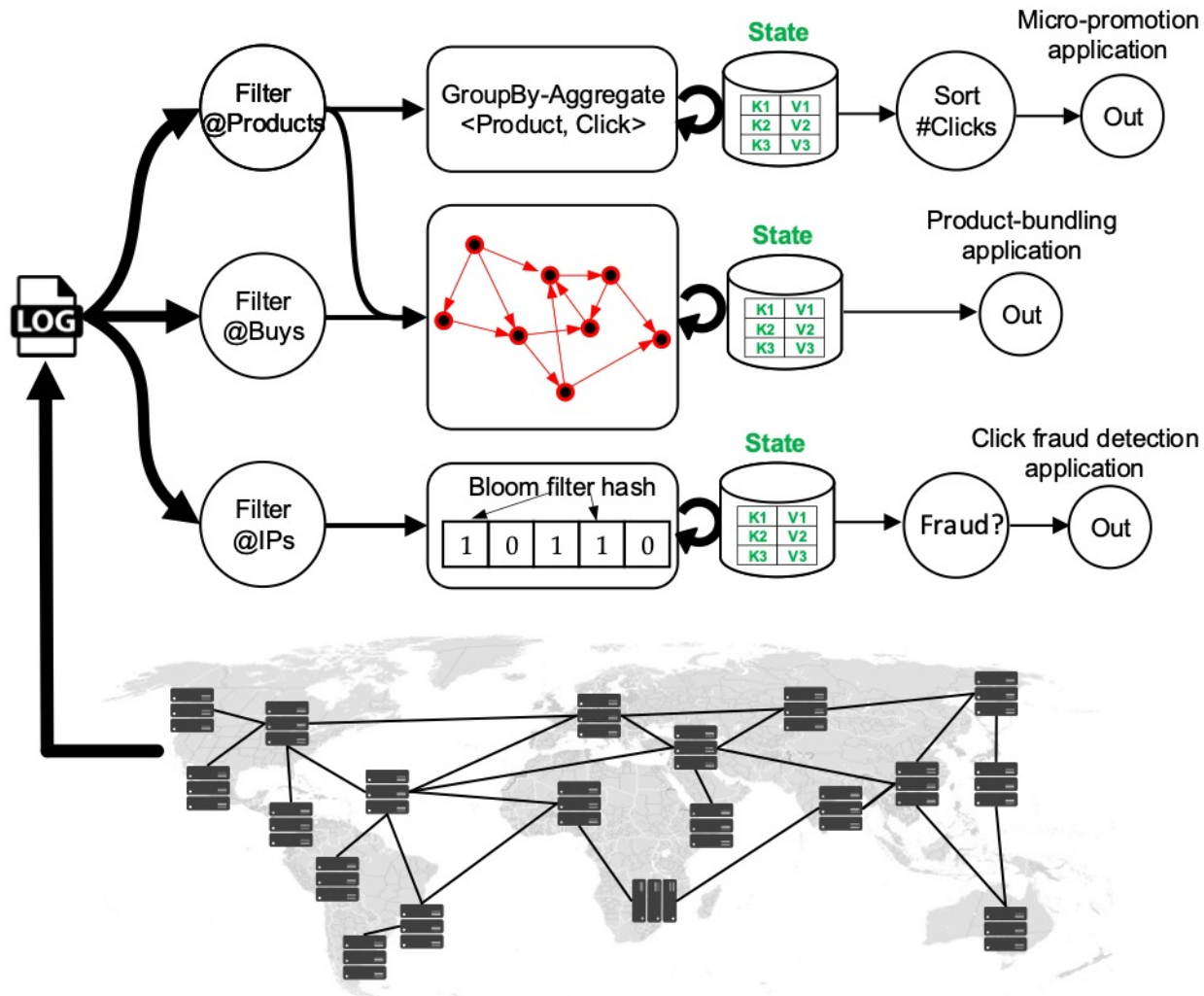
- Windows can be “tumbling” (5s window every 1s)



(1)



Background – An example



Background

- What if – state lost?
 - ▶ Need to be recovered

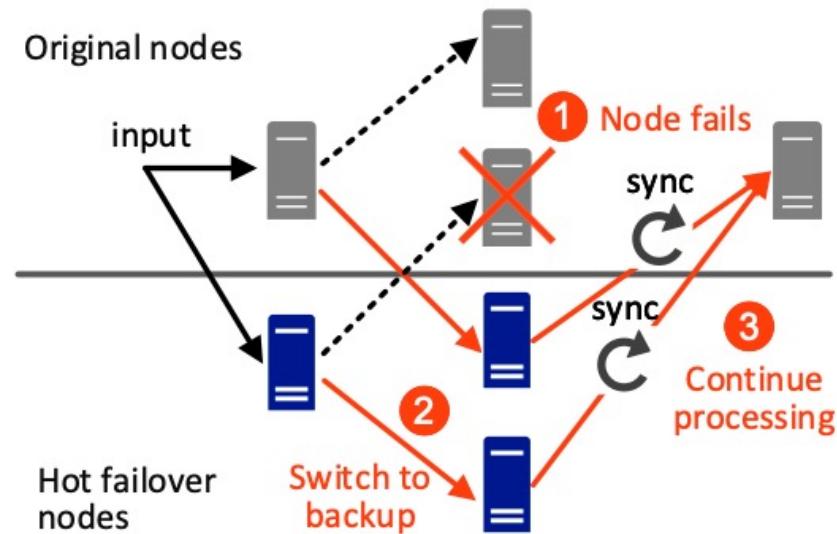


Figure: The replication recovery workflow

Background

- What if – state lost?
 - ▶ Need to be recovered

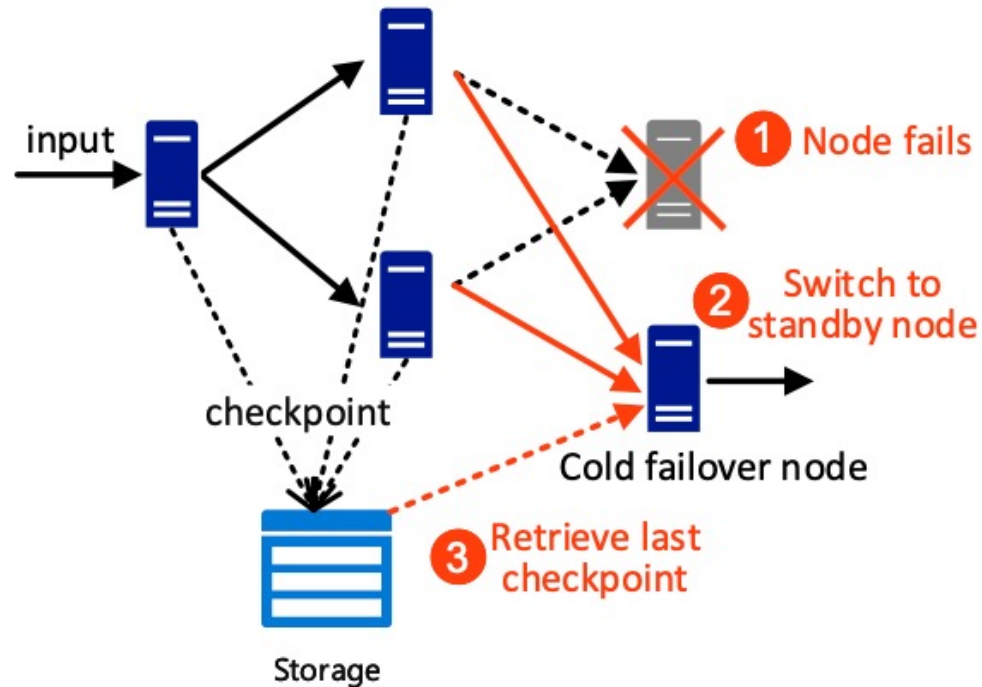


Figure: The checkpointing recovery workflow.

Background

- ❑ What if – state lost?
 - ▶ Need to be recovered

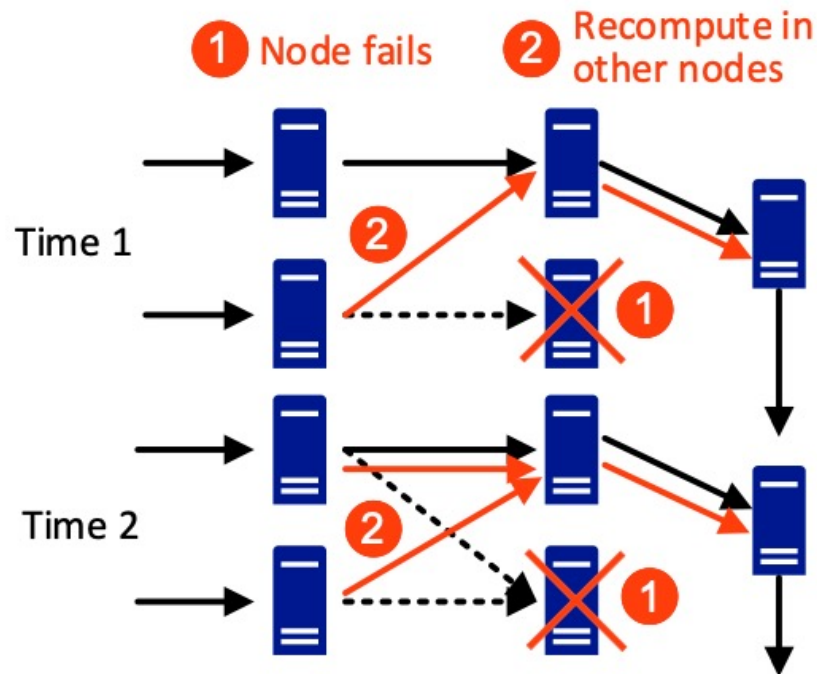


Figure: The DStream-based lineage recovery workflow


Background

Table 1. Overview of state management and state recovery solutions in stream processing systems.

System	Data Structure	State Management	State Recovery Approach	Scale to large state	Handle Multiple Failures	State Recovery Policy	State Recovery Traits
Muppet [26]	Slates [26]	In-memory	Checkpointing	×	×	Static	Slow
Trident [13]	Hashtable	In-memory	Checkpointing	×	×	Static	Slow
Millwheel [27]	Hashtable	Remote storage	Checkpointing	×	×		Slow
Dataflow [28]	Hashtable	Remote storage	Checkpointing	×	×	Static	Slow
Kafka [29]	Hashtable	In-memory+on-disk	Checkpointing	×	×	Static	Slow
Samza [30]	Hashtable	In-memory+on-disk	Checkpointing	×	×	Static	Slow
Flink [8]	Hashtable	In-memory+on-disk	Checkpointing	×	×	Static	Slow
Flux [19]	Hashtable	In-memory+on-disk	Replication	×	✓	Static	High cost
Borealis [20]	Hashtable	In-memory+on-disk	Replication	×	✓	Static	High cost
Spark Streaming [15]	RDDs [31]	In-memory+on-disk	DStream-based lineage recovery	×	✓	Static	Slow for long lineages

Challenges

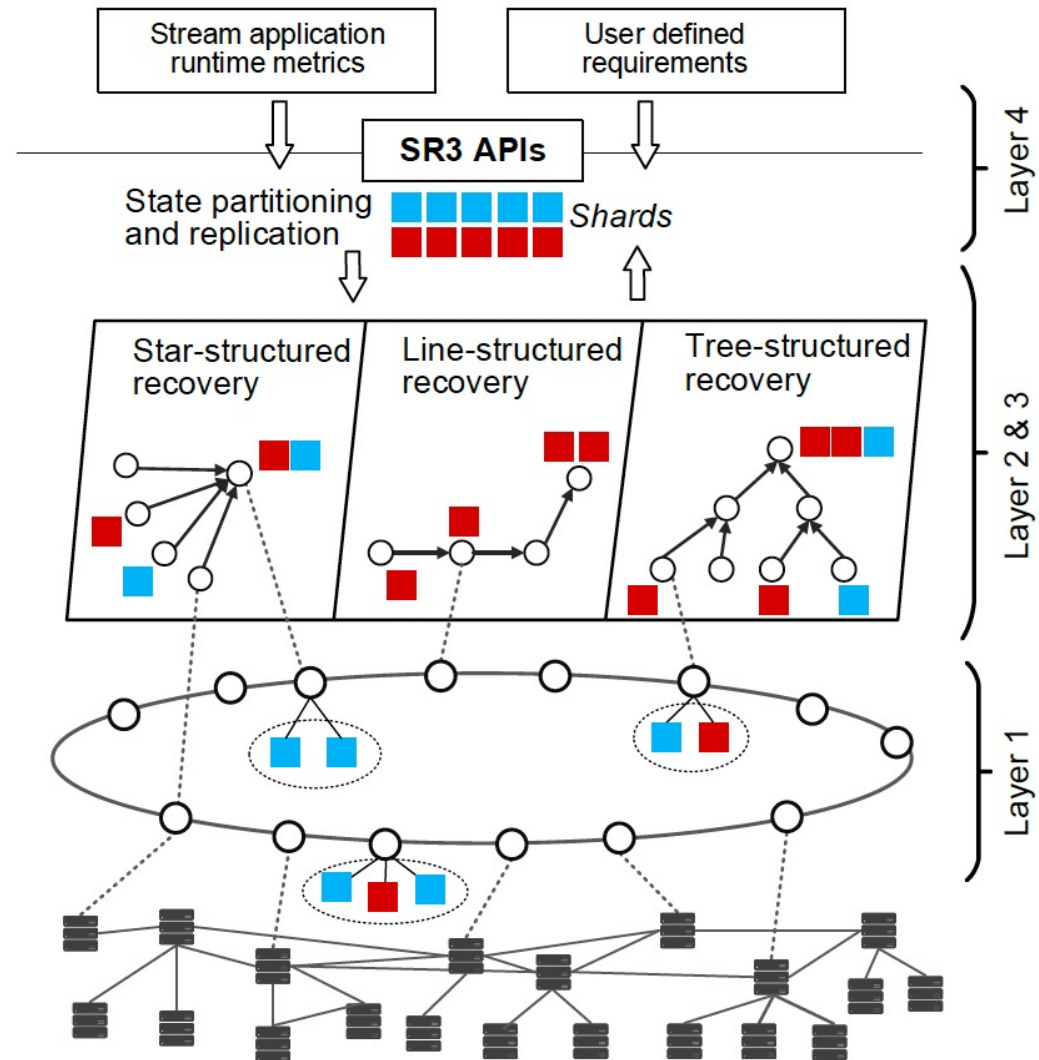
- ▶ Challenge 1: how to recover from **simultaneous** failures of **multiple** stream operators for many concurrently running applications?
- ▶ Challenge 2: how to customize the failure recovery mechanism for **different** types of stream processing applications?



SR3: a customizable state recovery framework
that provides fast and scalable failure recovery
mechanisms

System Design

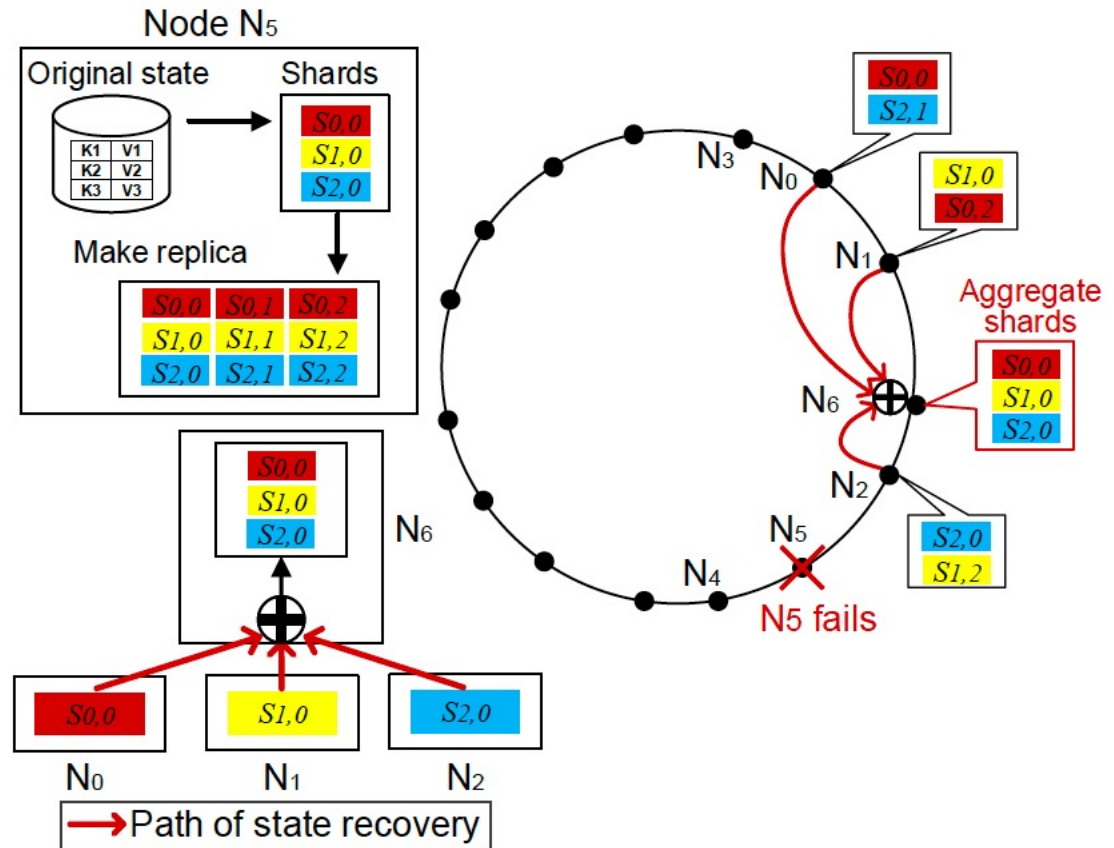
- ▶ L1: DHT-based overlay.
- ▶ L2: State partitioning and replication.
- ▶ L3: State recovery.
- ▶ L4: SR3 API.



System Design – Star-structured

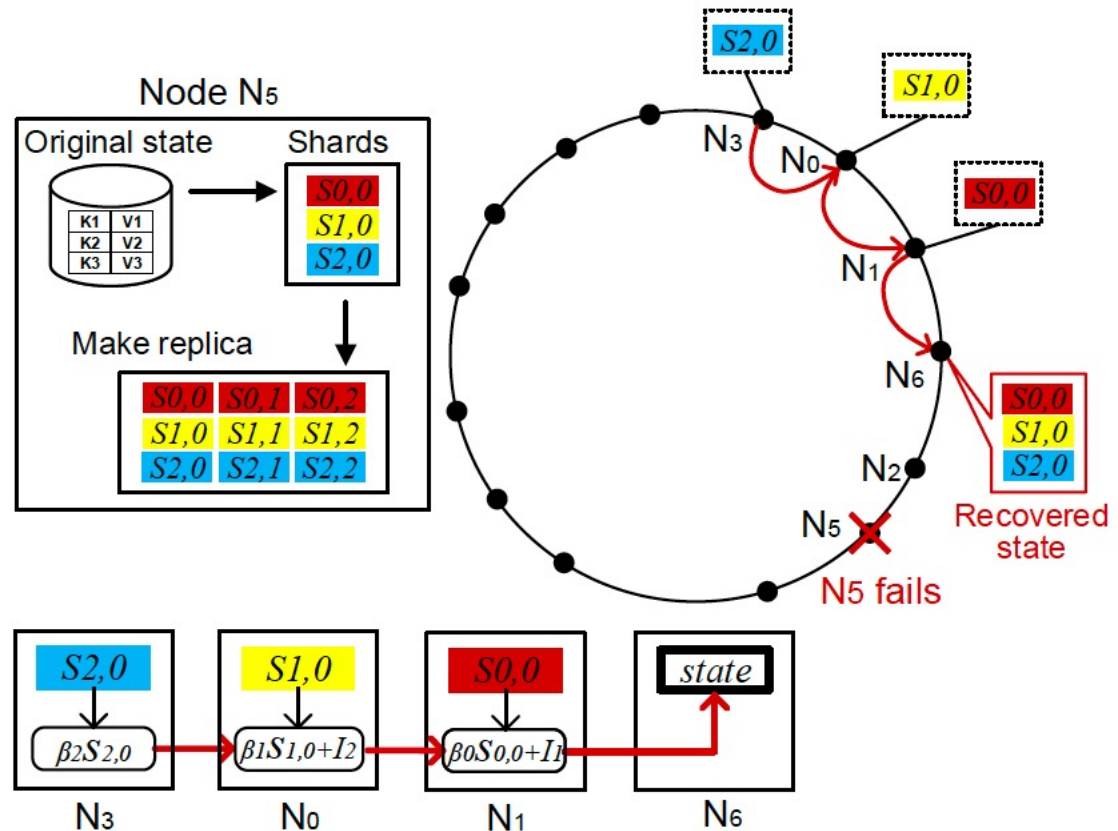
Benefits:

- ▶ The recovery process is fast.
- ▶ Achieve data locality because the leaf set contains nodes that are geographically close to the original node.



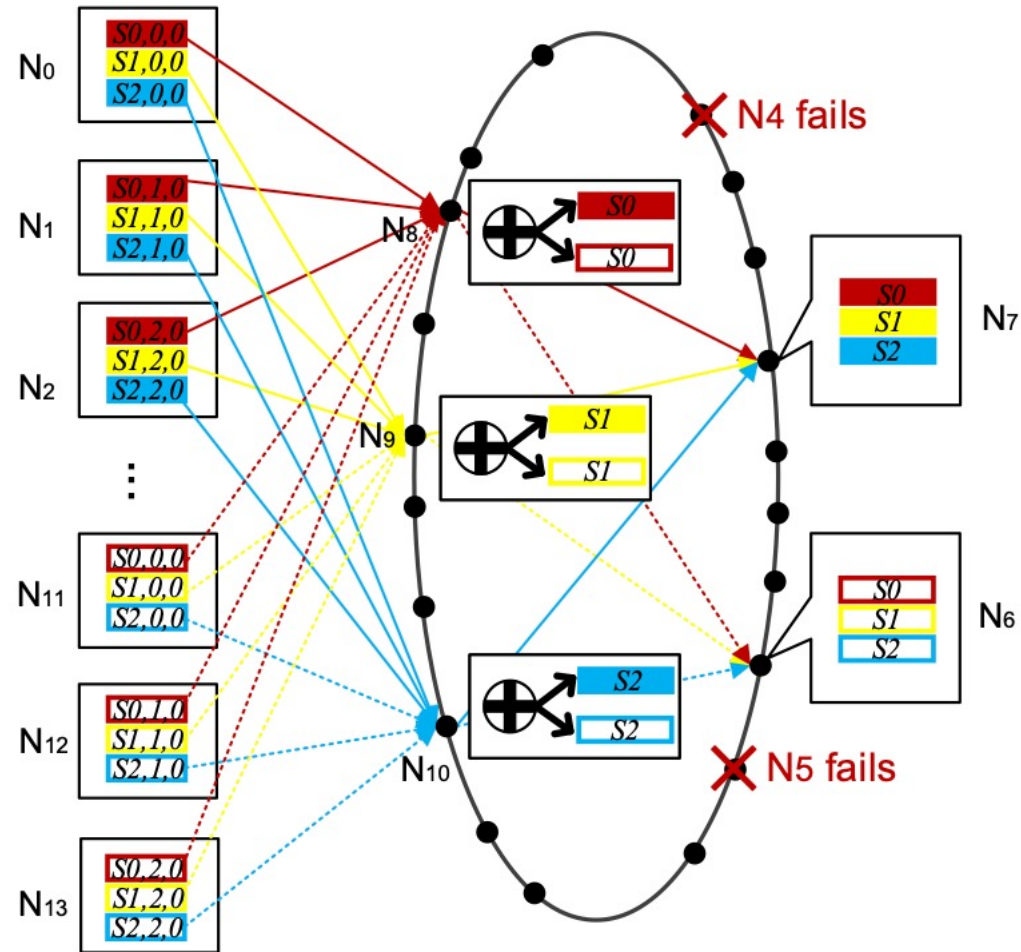
System Design – Line-structured

- ▶ Benefits:
 - ▶ The downloading and computing load are well balanced among all involved nodes.
 - ▶ Avoid centralized bottleneck.

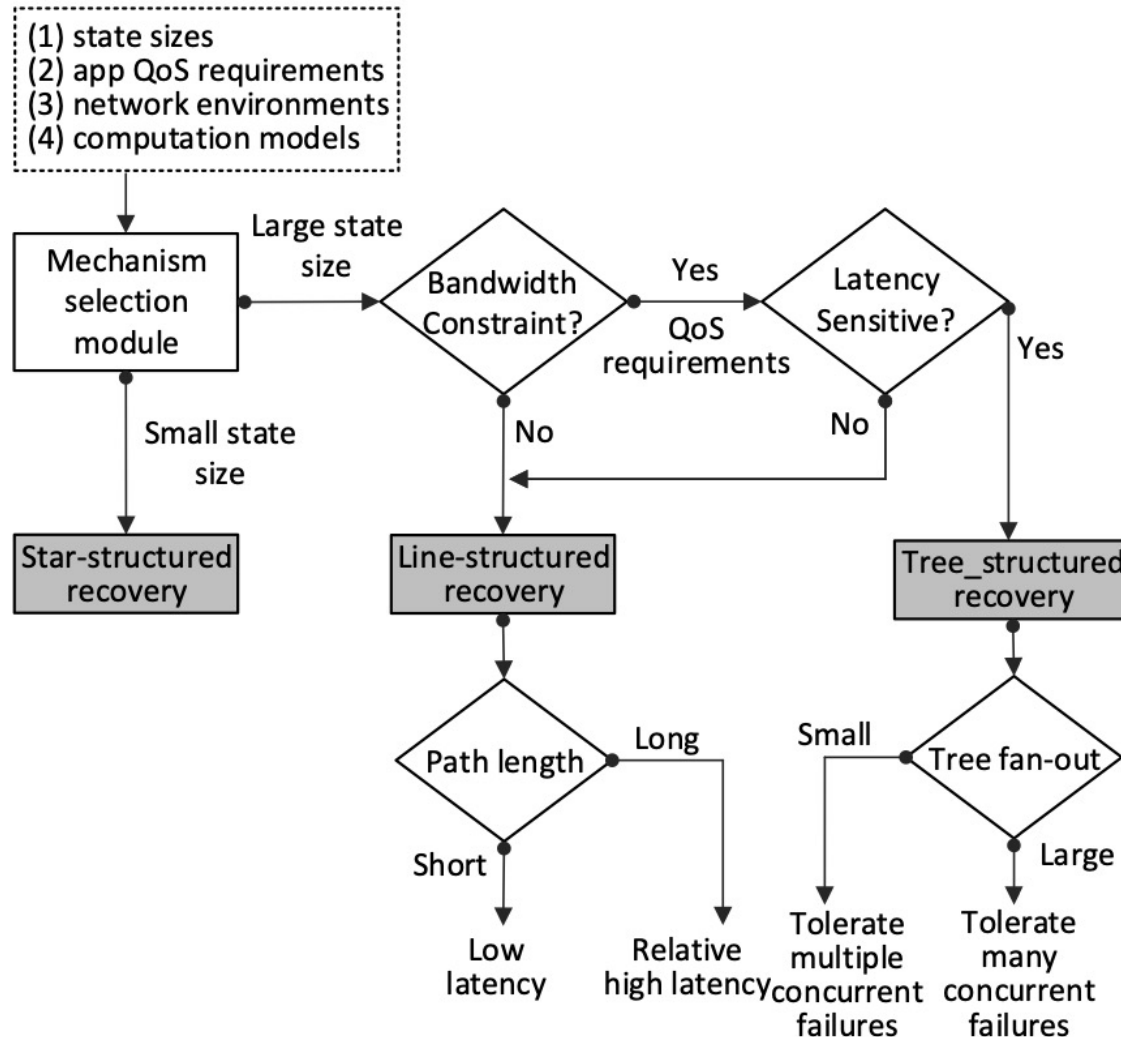


System Design – Tree-structured

- ▶ The spanning tree is built on top of a scalable application-level multicast infrastructure.
- ▶ Benefits:
 - ▶ A providing node only needs to upload some of the shards it stores.
 - ▶ The downloading and computing load are well balanced among all involved nodes



System Design – Mechanism selection



Implementation

- We implement the SR3 system on top of Apache Strom (v.2.0.0) and Pastry (v.2.1) software stacks.
- SR3 APIs

List<> StateSplit(String state, Integer numberOfShards, Integer numberOfReplicas)

Boolean[] Save(List<>shard, DHTNetwork overlay)

Boolean StarDefine(String appName, Integer starFanout)
--

Boolean LineDefine(String appName, Integer lengthofPath)
--

Boolean TreeDefine(String appName, Integer fanout, Integer branchDepth)

Integer Selection(String appName, String requirement, Long state-Size, Long networkBW)
--

List<Shards>Recover(String stateName, DHTNetwork overlay, Mechanism structure)
--

- ❑ Our evaluation answers the following questions:
 - ▶ Does SR3 improve the state save and recovery performance when deploying different stream applications with various sizes of states?
 - ▶ Does SR3 support flexible state recovery in handling various sizes of states with different network environments?
 - ▶ Does SR3 scale with the number of concurrently running stream applications?
 - ▶ What is the runtime overhead of SR3?

Evaluation

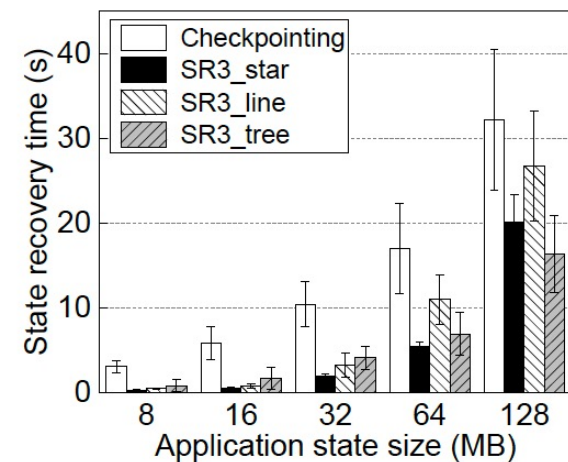
- ❑ Testbed of 50 virtual machines (VMs) running Linux 3.10.0,
- ❑ Each virtual machine has 4 cores and 8GB of RAM, and 60GB disk.
- ❑ One JVM to emulate an SR3 node and emulate up to totally 5,000 SR3 nodes in our testbed
- ❑ Baseline: Apache Storm 2.0.0 configured with 10 TaskManagers, each with 4 slots
- ❑ Pastry 2.1 with leaf set size of 24 and transport buffer size of 32MB

Table 3. Real-world application's dataset.

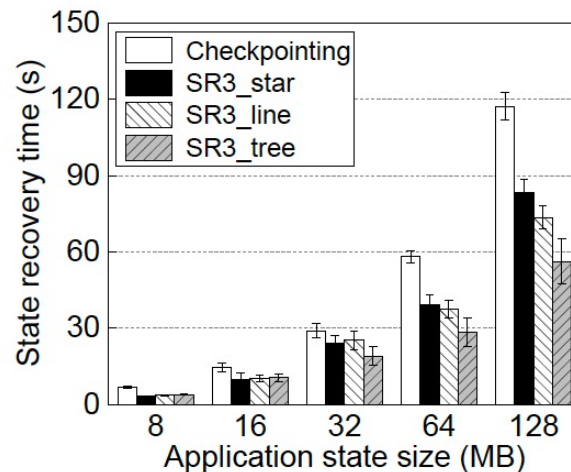
Application	Dataset	Size
Bargain Index	Google Finance [55]	>1TB
Word Count	Wikimedia Dumps [56]	9GB
Traffic Monitoring	Dublin Bus Traces [57]	4GB

Evaluation

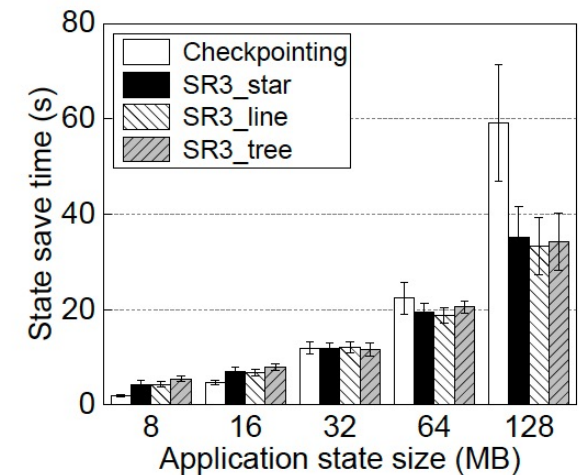
- The time of state save and recovery in checkpointing, star-structured recovery, line-structured recovery, and tree-structured recovery.
- SR3 generally achieves 35.5% ~ 65% less state recovery time.



(a) The state recovery time by varying the size of state with no bandwidth constraint.



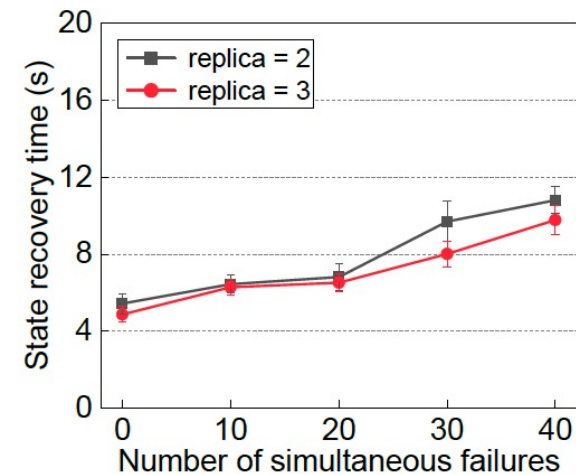
(b) The state recovery time by varying the size of state with bandwidth constraint.



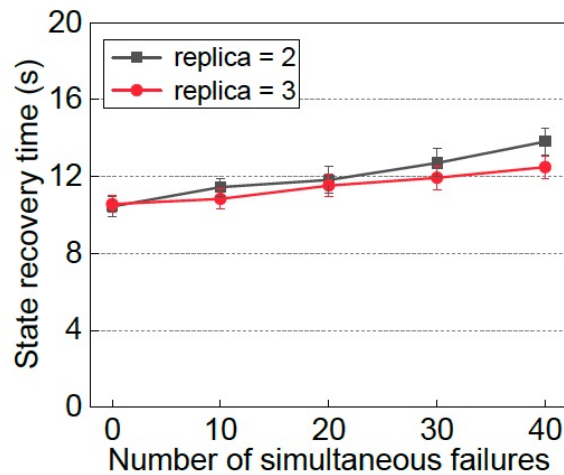
(c) State save time by varying the size of state.

Evaluation

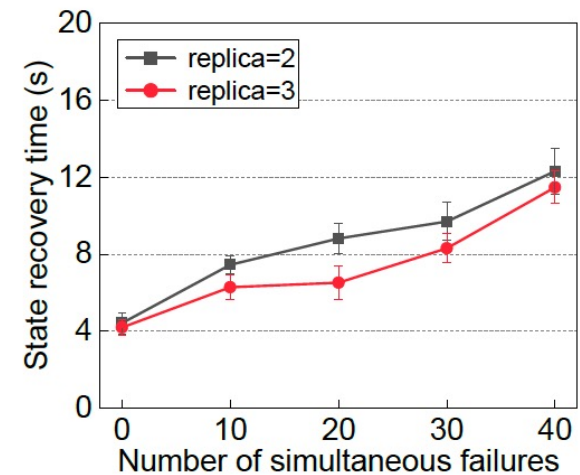
- State recovery time with different number of failures.
 - ▶ To cause simultaneous failures, we deliberately remove some shards of application's state in some nodes.
 - ▶ Their two curves show that the recovery time slightly increase with increasing number of shards failures.



(a) The state recovery time with failures in SR3 star-structured recovery.



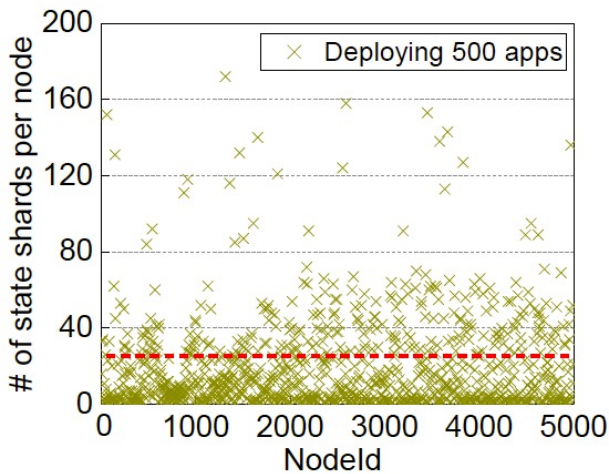
(b) The state recovery time with failures in the SR3 line-structured recovery.



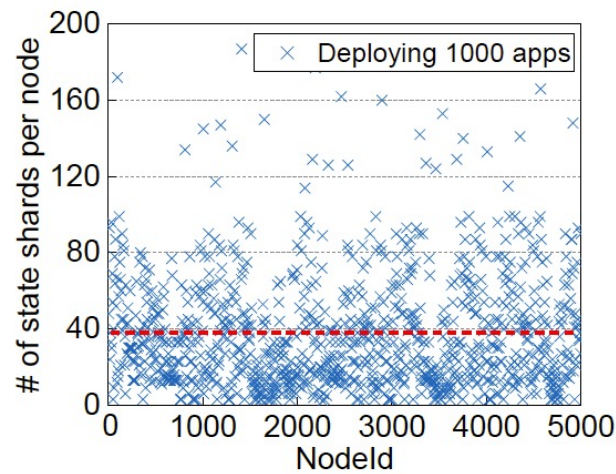
(c) The state recovery time with failures in SR3 tree-structured recovery.

Evaluation

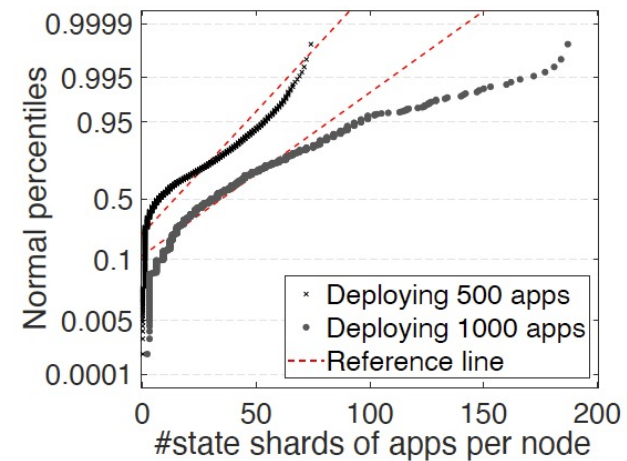
- ❑ Performance evaluation of SR3 in terms of load balance and scalability.
 - SR3 has attractive load balance property because it distributes state across all nodes in the overlay, which is especially beneficial when deploying many concurrent applications.



(a) The distribution of state among the overlay when deploying 500 applications.



(b) The distribution of state among the overlay when deploying 1,000 applications.

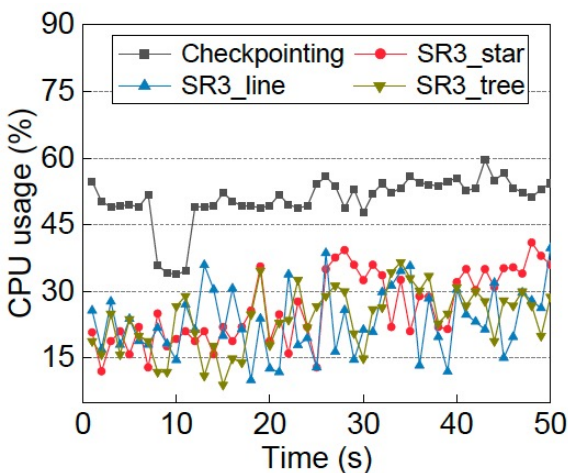


(c) Normal probability of the number of shards per node.

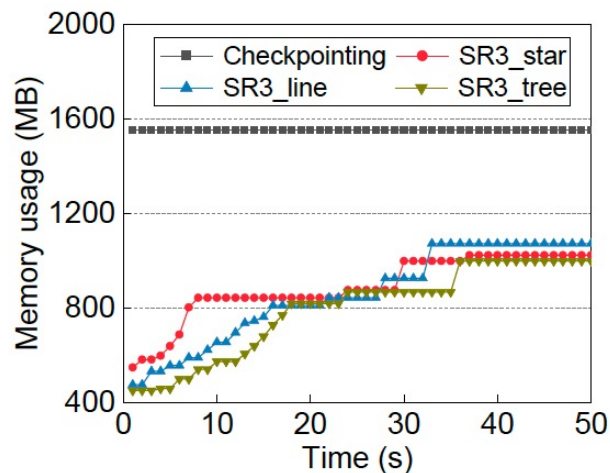
Evaluation

□ Evaluate SR3 runtime overhead

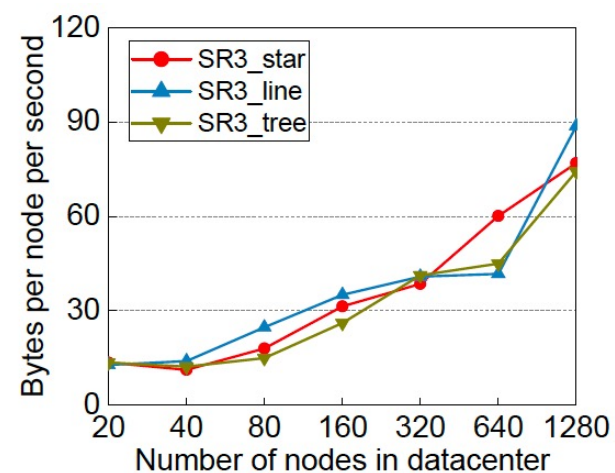
- ▶ Compare with the checkpointing approach having less CPU and memory cost.
- ▶ The network cost mainly comes from the ping-pong messages that maintaining the DHT network.



(a) The CPU overhead.



(b) The memory overhead.



(c) The network traffic overhead per node.

Conclusion

- ▶ First, we show how existing techniques can lead to slow or resource expensive state recovery that is not scalable and identify the causes of their shortcomings.
- ▶ Second, we propose SR3, a customizable state recovery framework that provides fast and scalable failure recovery mechanisms for protecting large distributed states in stream processing systems.
- ▶ Third, SR3 provides three different failure recovery mechanisms.
- ▶ Fourth, we present the integration of SR3 onto the Apache Storm framework.
- ▶ Finally, we make a comprehensive evaluation of the scalability, fast recovery and flexibility of SR3.



Thank you !

Contacts: hailu.xu@csulb.edu