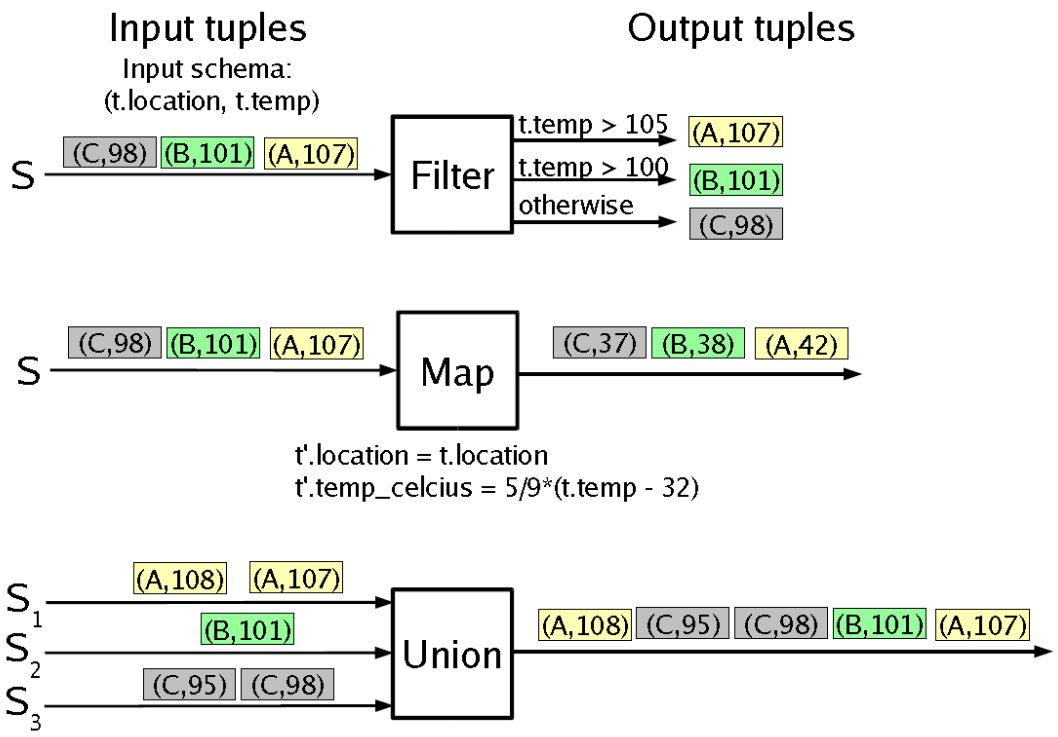
Infrastructure Technologies for Large-Scale Service-Oriented Systems

Kostas Magoutis
magoutis@cse.uoi.gr
http://www.cse.uoi.gr/~magoutis

Samza

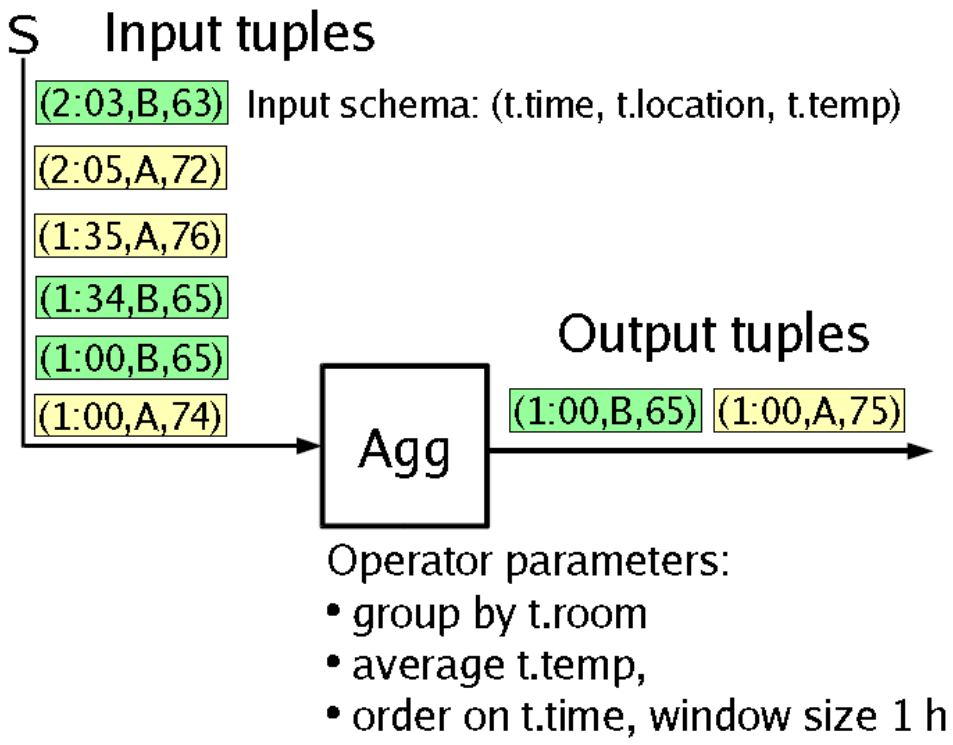
- Efficient support for state
- Fast failure recovery and job restart
- Reprocessing and lambda-less architecture
- Scalability

General principles: Stateless operators



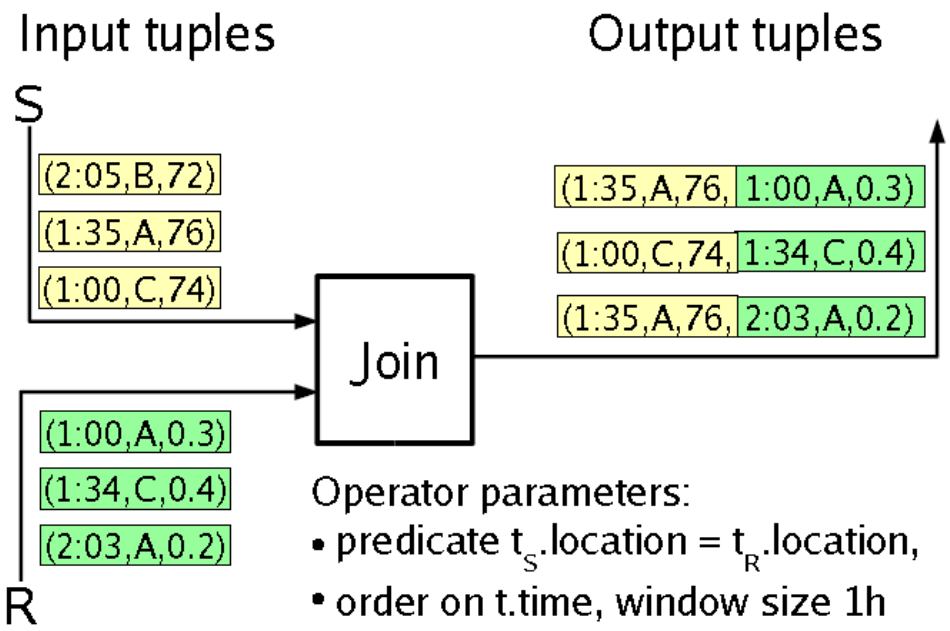
Courtesy of Borealis Application Programmer's Guide, Brown Univ. Computer Science Department

General principles: Aggregate operator



Courtesy of Borealis Application Programmer's Guide, Brown Univ. Computer Science Department

General principles: Join operator

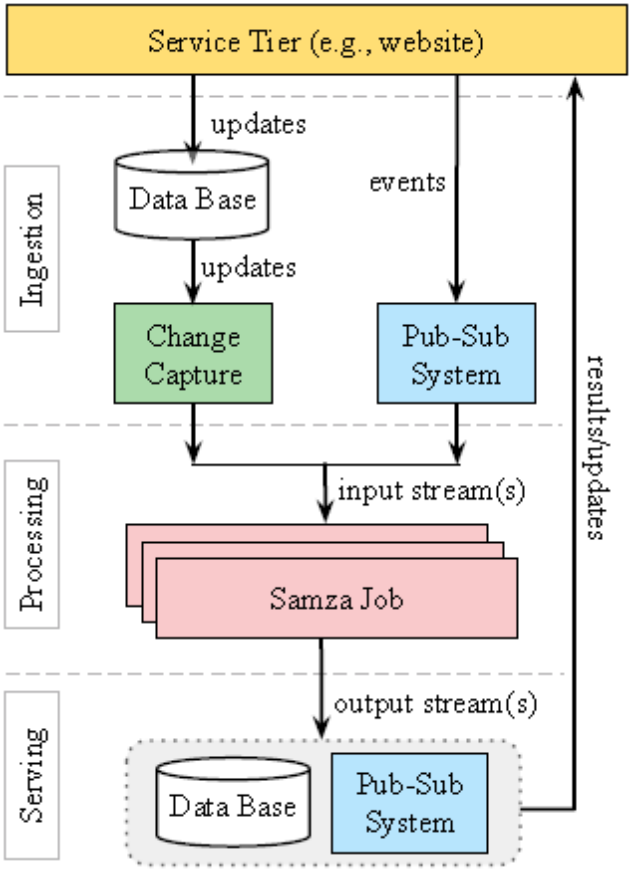


Courtesy of Borealis Application Programmer's Guide, Brown Univ. Computer Science Department

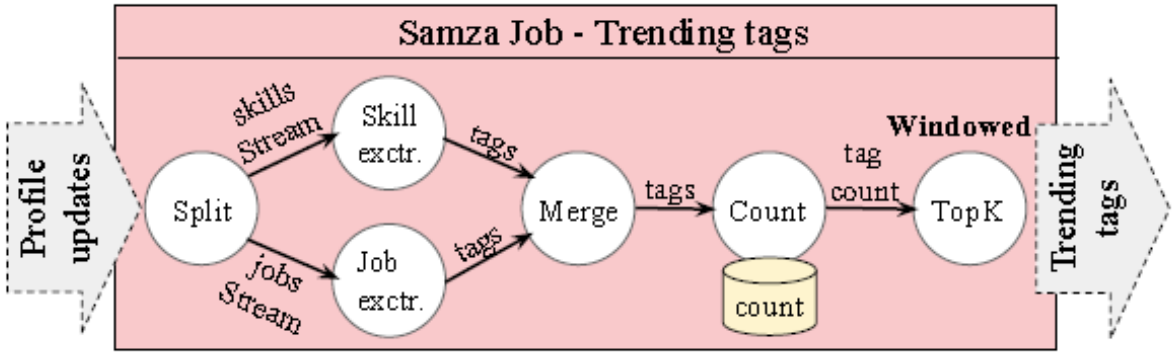
Stateful processing

Email digestion system (read-only, read-write state)

Stream processing pipeline at LinkedIn



Samza job to find trending tags



Type	Options	Definition
1:1	map	applying a defined function on each message.
	filter	filtering messages based on a function.
	window	splitting a stream into windows and aggregating elements in the window.
	partition	repartitioning a stream on a different key.
m:1	join	joining $\geq$ 2 streams into one stream based on a given function
	merge	merging $\geq$ 2 two streams into one stream.
1:m	user-defined	user-defined split or replication of a stream into $\geq$ 2 streams. This is achieved by allowing multiple operators consume the same stream.

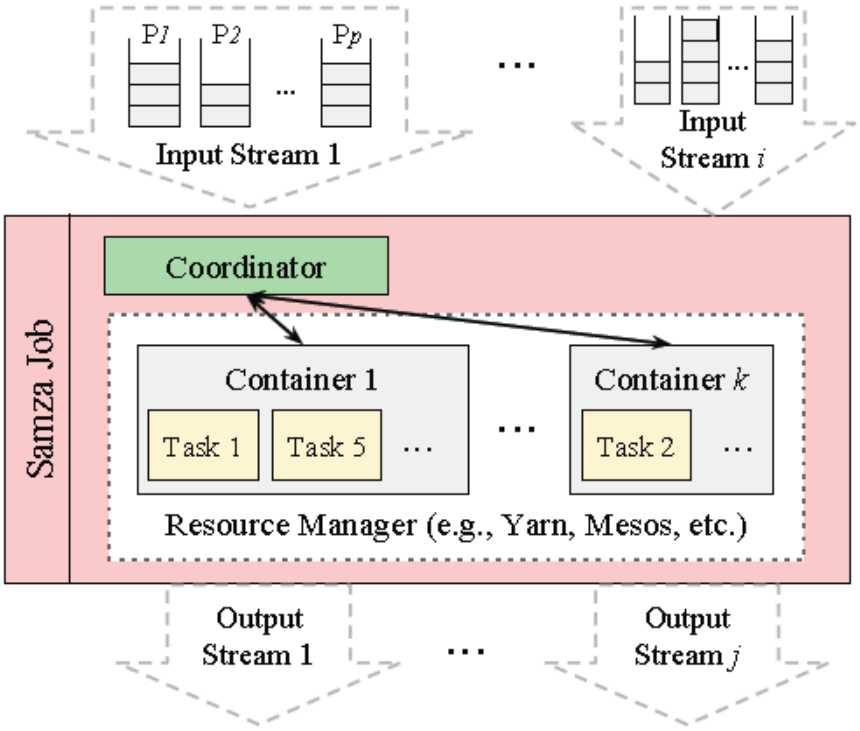
Trending tags job

```
public void create(StreamGraph graph, Config conf) {
   //initialize the graph
   graph = StreamGraph.fromConfig(conf);
   MsgStream<> updates = graph.createInStream();
   OutputStream<> topTags = graph.createOutStream();

   //create and connect operators
   MsgStream skillTags = updates.filter(SkillFilter f_s)
                                .map(SkillTagExtractor e_s);
   MsgStream JobTags = updates.filter(JobFilter f_j)
                                .map(JobTagExtractor e_j);
   skillTags.merge(jobTags).map(MyCounter)
         .window(10, TopKFinder).sendto(topTags);
         //10 sec window
}

class MyCounter implements Map<In, Out>{
//state definition
 Store<String, int> counts = new Store();
 public Out apply (In msg){
    int cur = counts.get(msg.id) + 1;
    counts.put(msg.id, cur);
    return new Out(msg.id, cur)
 }
}
```

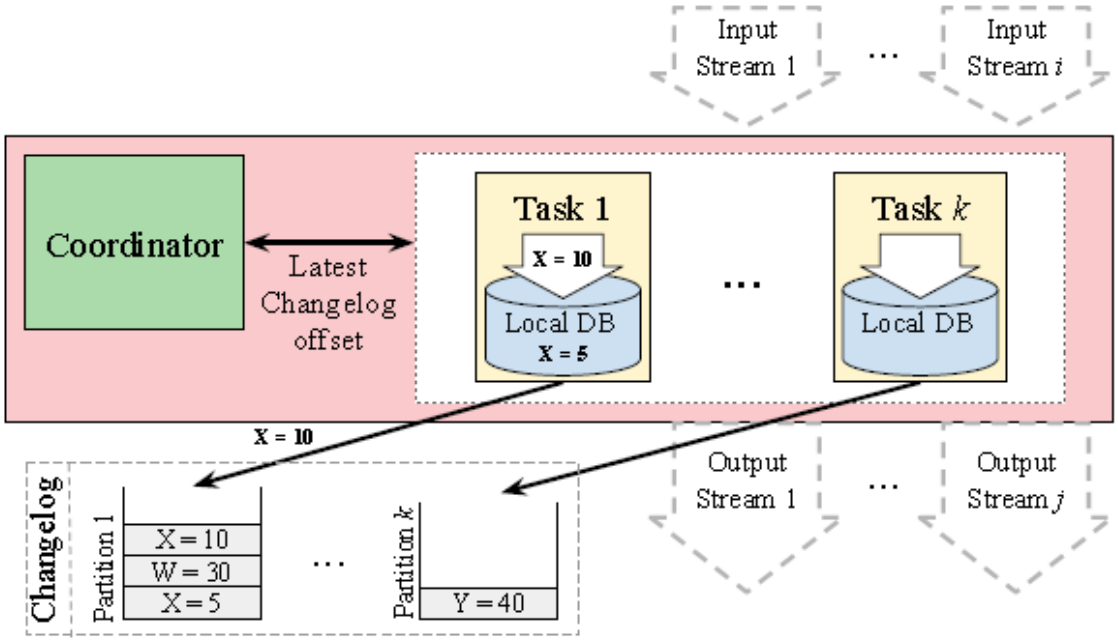
Internal structure of a job



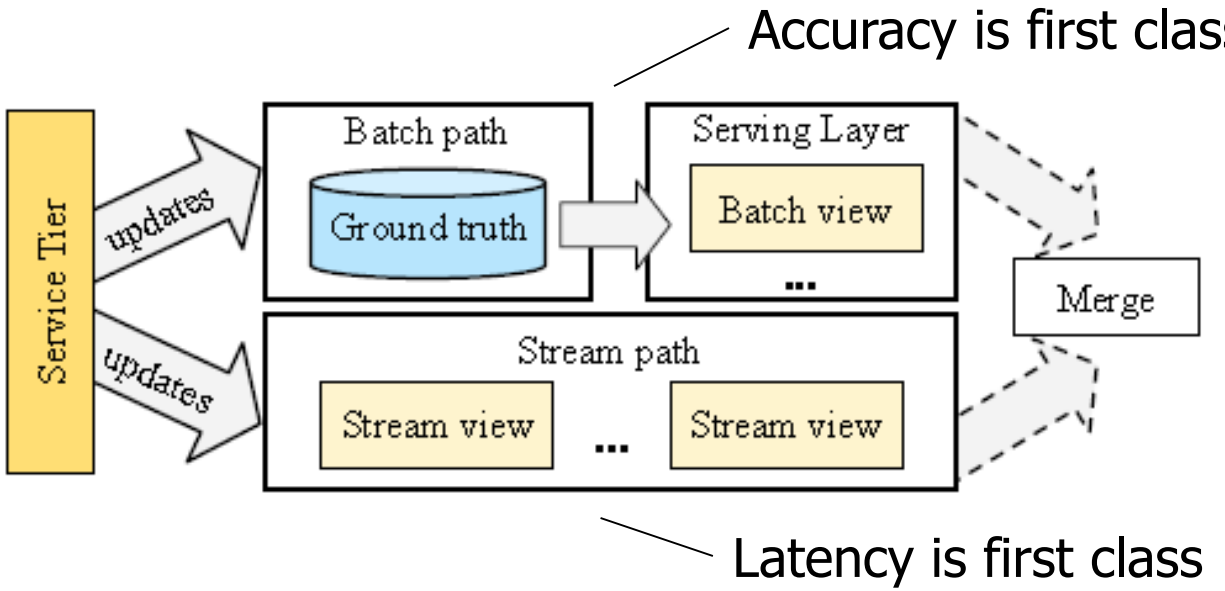
Samza-based applications at LinkedIn

Name	Definition	Containers	Tasks	Inputs	Throughput msg/s	State type
EDS	Digesting updates into one email (aggregation, look-up, and join).	350	2500	14	40 K	on-disk
Call graph	Generating the graph of the route a service call traverses (aggregation).	150	**9500**	620	**1 Million**	in-mem
Inception	Extracting exception information from error logs (stateless filter).	300	300	**880**	700 K	stateless
Exception Tracing	Enriching exceptions with the source (machine) of the exception (join).	150	450	5	150 K	in-mem
Data Popularity	Calculating the top k most relevant categories of data items (join and machine learning).	70	420	9	3.5 K	on-disk
Data Enriching	Enriching the stream of data items with more detailed information (join).	350	700	2	100 K	on-disk
Site Speed	Computing site speed metrics (such as average and percentiles) from the stream of monitoring events over a 5-minute window (aggregation).	350	600	2	60 K	in-mem
A/B testing	Measuring the impact of a new feature. This application first categorizes input data (by their tag) into new and old versions and then computes various metrics for each category (split and aggregate).	450	900	2	100 K	in-mem
Standardization (>15 jobs)	Standardizing profile updates using machine learning models. This application includes > 15 jobs, each processing a distinct features such as title, gender, and company (join, look-up, machine learning).	**550**	5500	3	60 K	in-mem remote on-disk

Layout of local state – fault tolerance



Lambda architecture



Samza is lambda-less:

- Unified model: treats batch data as finite data stream
- Processing late events: Avoid reprocessing entire stream
- Reprocessing: leverage Kafka/Databus replaying capability
  - Block real-time computation until reprocessing complete
  - Reprocess in parallel with real-time processing

Scalable design

- Scaling resources
  - Job split into independent and identical tasks
  - Input/state partitioning
  - Tasks allocated on containers, can be migrated
- Scaling state
  - Leverage independent partitioned local stores
  - Recovery in parallel across tasks
- Scaling input sources
  - Treat each input stream autonomously from other inputs
  - Works with variety of systems
- Scaling number of jobs
  - No system-wide master
  - Jobs are independent, placed on their own set of containers

Checkpointing vs changelog

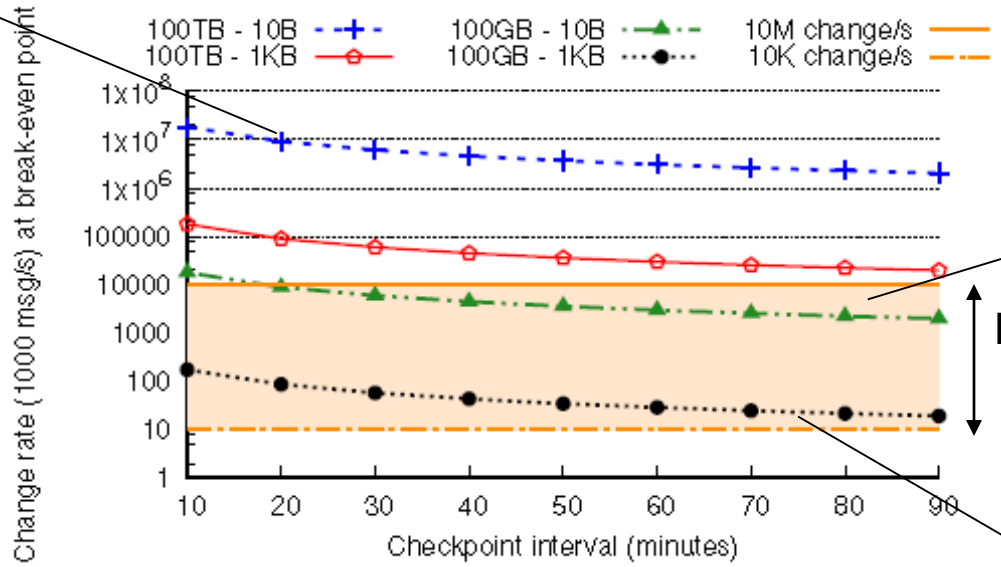
Approach	Parameter	Definition	Range
Checkpoint	*interval*	time between two consecutive checkpoints.	10 min - 90 min
	state_size	total size of state in Bytes	100 GB - 100 TB
Changelog	*change_rate*	rate of entry changes in the state (msg/s).	10 K - 10 M
	entry_size	size of each entry of the state in Bytes	10 B - 1 KB

$$Data_{checkpoint} = \frac{state\_size}{interval}$$

$$Data_{changelog} = change\_rate \times entry\_size$$

Checkpointing vs changelog

10 trillion changes/sec



Changelog worse than checkpointing

Realistic estimate for change rate

Uncommon in practice

$$Data_{checkpoint} = \frac{state\_size}{interval}$$

$$Data_{changelog} = change\_rate \times entry\_size$$

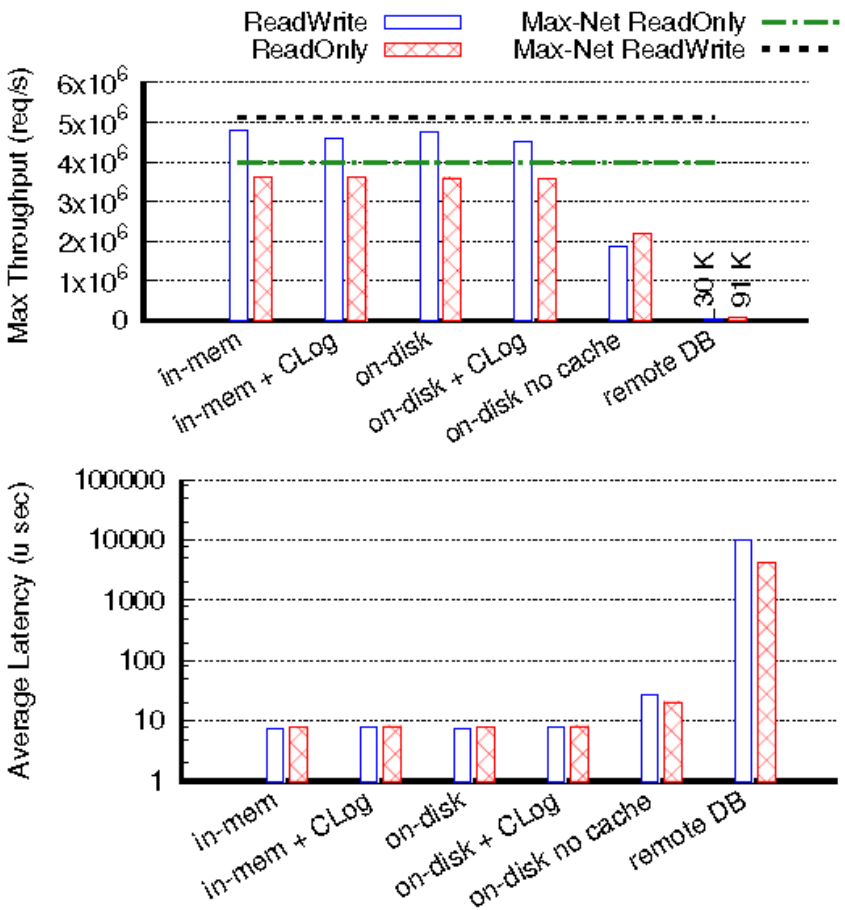
Experimental setup

- Small (6-node) and large (500-node) clusters
- ReadOnly job (Data enriching, Exception tracing)
  - Join between a database and an input stream
    - Extract embedded id from each message
    - Read (id, val) from database
    - Join val with input message
    - Output result as new message
- ReadWrite job (EDS, Call graph, Site speed)
  - Map ids to counters:
    - Extract embedded id from each message
    - Read count for id
    - Increment counter
    - Write counter back

Experimental setup

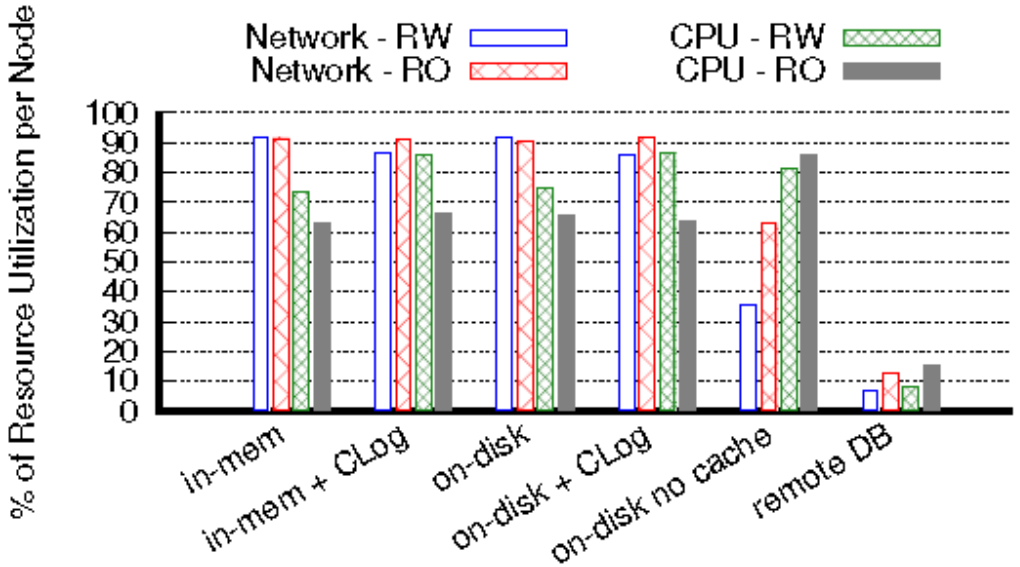
- Single input stream
- Infinite tuples (id, padding)
  - id is randomly generated in range [1, 10^k]
  - padding is randomly generated string of size m
- k and m are tuning knobs
  - k trades off state size for locality
  - m is used to tune CPU/network usage
- Choose m such that the system is under stress
  - 100 bytes for ReadWrite, 130 bytes for ReadOnly

In-mem vs local disk vs remote disk

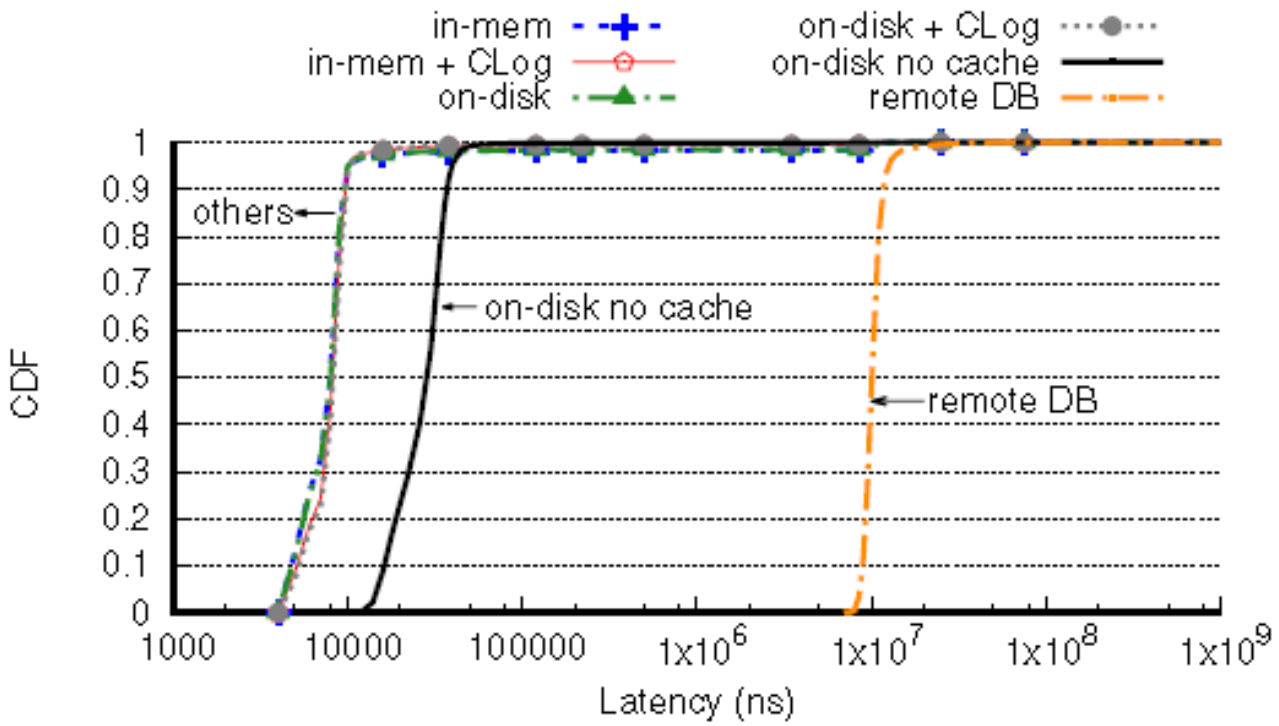


Network b/w divided by message size

Network (inbound), CPU utilization



Latency


in-mem
in-mem + CLog
on-disk
on-disk + CLog
on-disk no cache
remote DB
CDF
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
others
on-disk no cache
remote DB
1000
10000
100000
1x10^6
1x10^7
1x10^8
1x10^9
Latency (ns)

Failure recovery