

A photograph of several cosmos flowers in shades of pink and magenta with yellow centers, set against a blurred green background. The top portion of the image is overlaid with a blue sky and a semi-transparent white grid pattern where the title text is located.

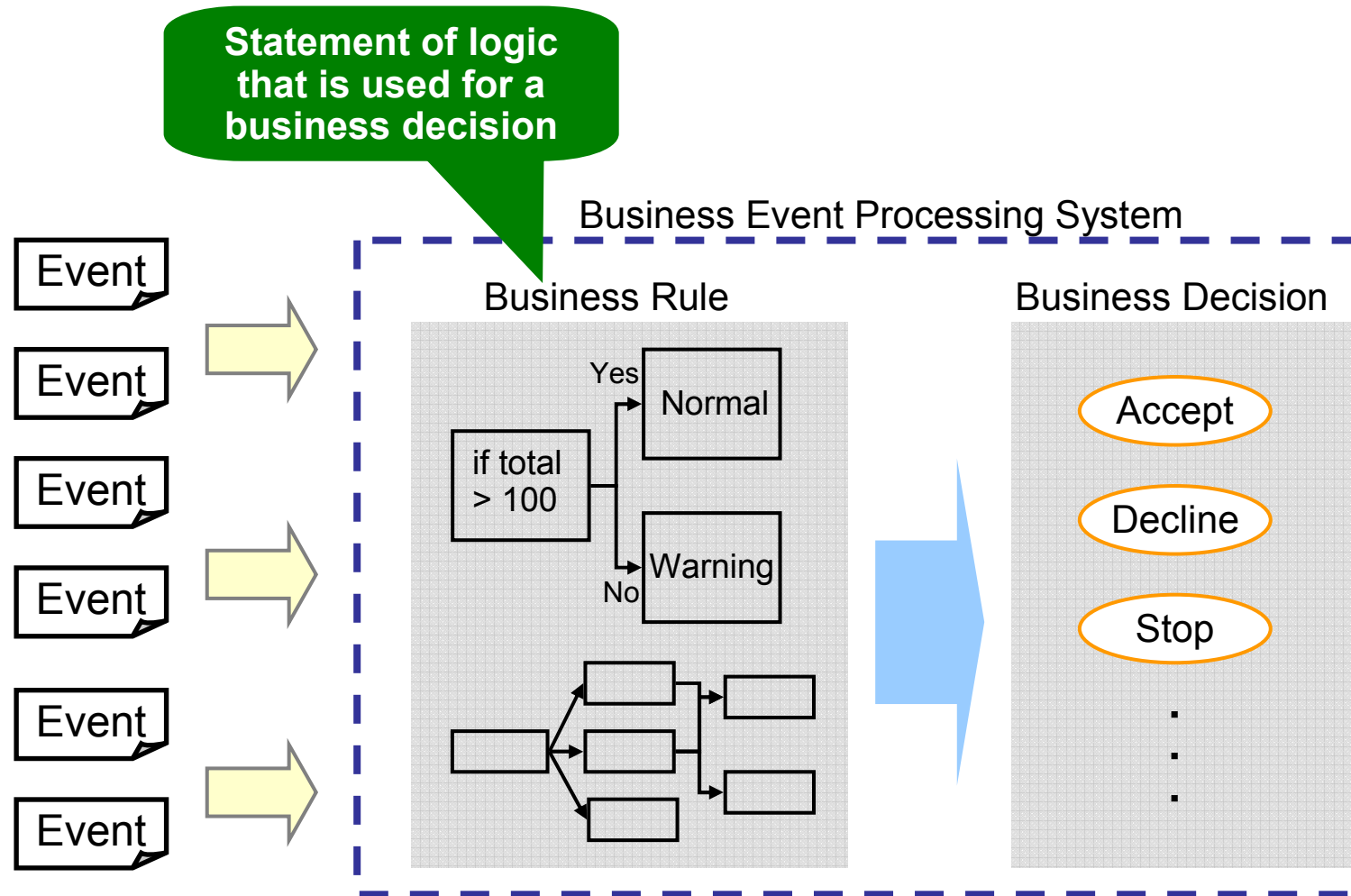
Event Processing over a Distributed JSON Store - Design and Performance -

IBM Research

Miki Enoki, Jerome Simeon
Hiroshi Horii, Martin Hirzel

Business Event Processing

- Event-driven architecture to coordinate human activities along with automated tasks

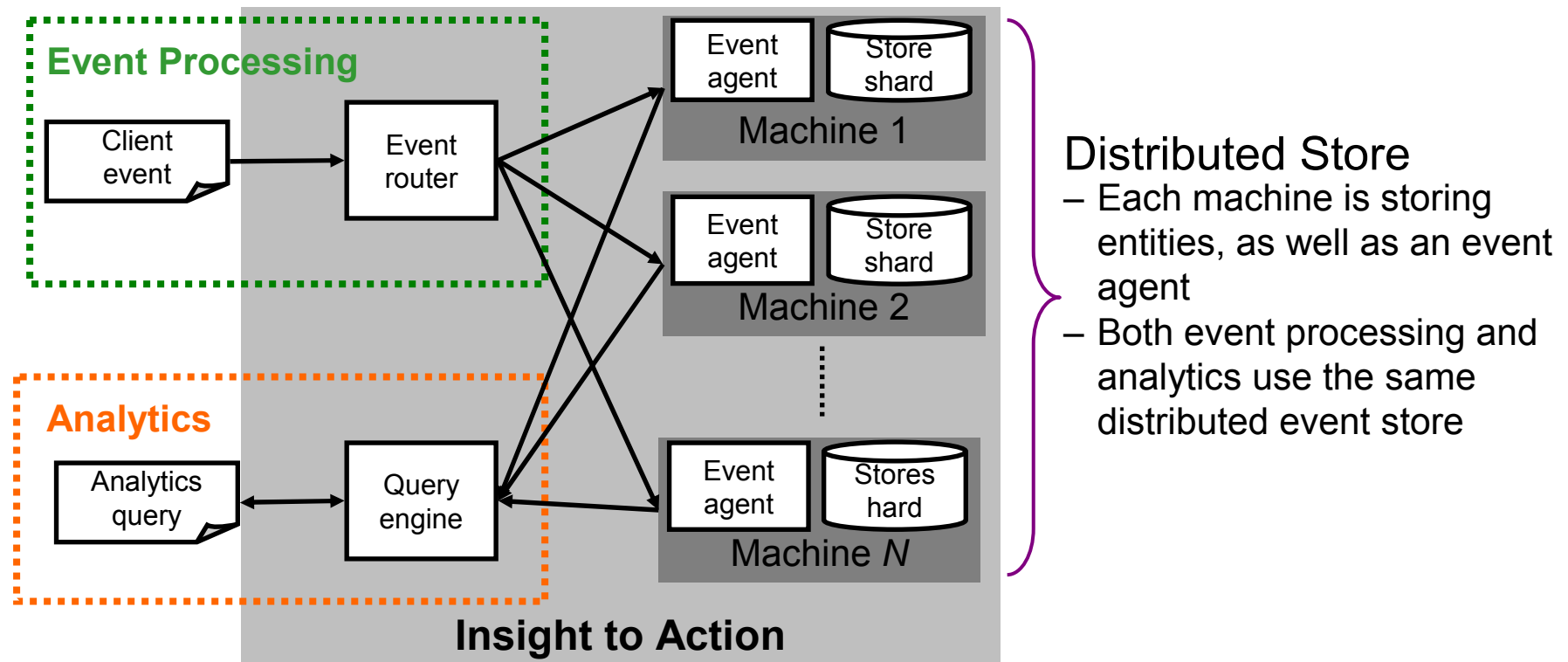


Our Contributions

- Describe a system that supports combined workloads for **events** and **analytics** over an in-memory distributed JSON store
- Introduce the architecture for the distributed JSON store, which leverages the MongoDB API, and show experimental results on throughput, latency, and scalability

Insight to Action(I2A) combines event processing with analytics over a distributed store

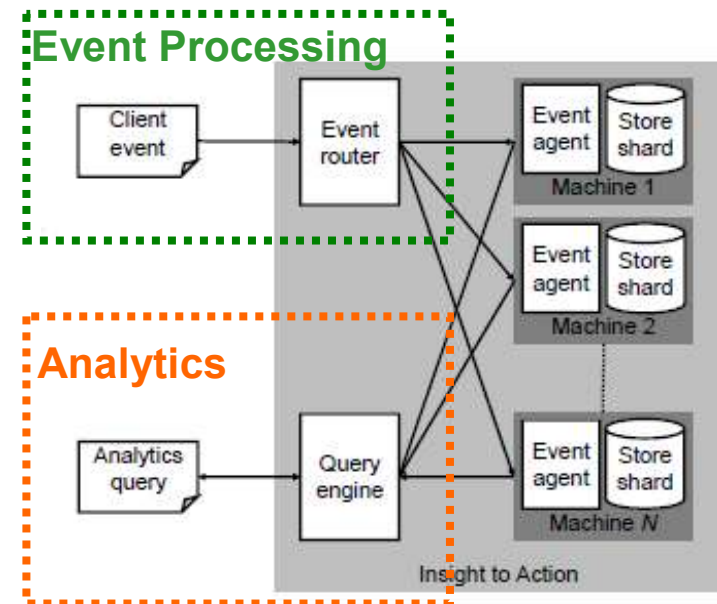
- I2A is under development at IBM
- Combining both events and analytics in a single system fosters
 - ease of use (no need to configure multiple systems)
 - performance (no need to move data back and forth)



Event Processing in I2A

■ Event Processing

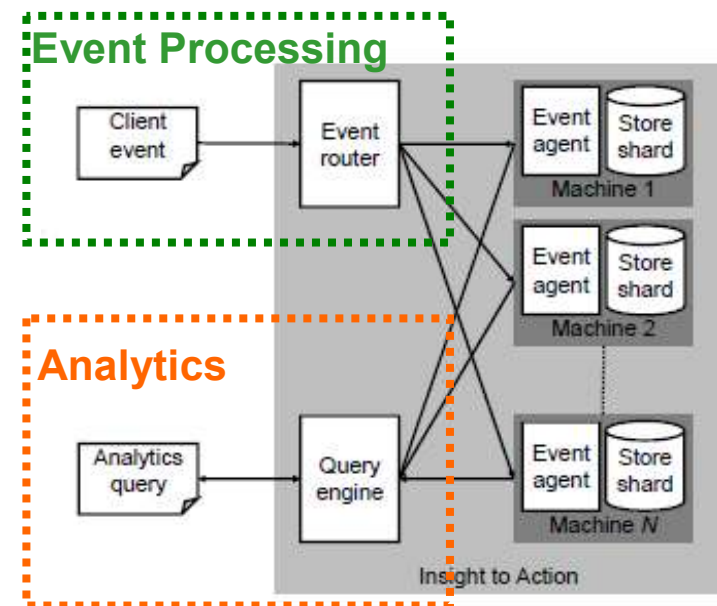
- Each arriving client event contains a key associating it with an entity
- The system uses the key in the event to route it to the machine where the corresponding entity is stored
- The event agent acts upon the event by reading and writing entities in the store and emitting derived events



Analytics in I2A

■ Analytics

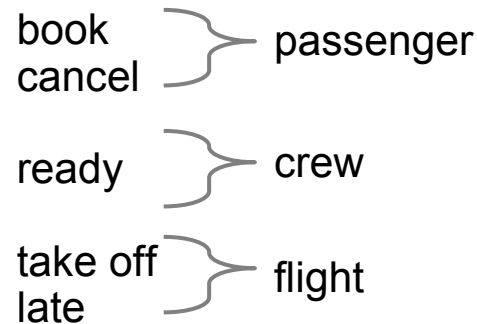
- Analytics can be either user-initiated, or scheduled to repeat periodically
- The query engine coordinates the distributed analytics, and combines the results (i.e., insight)
- The results are then either reported back to the user, or saved in the store for use by future events (i.e., action)



Example Scenario : Airline Industry – Event Processing

Event

Entity



- Event agent handles incoming events and applies business rules to them
- If rule condition is matched, the agent can update the entity and/or emit new events

Business Rule

- If it receives take off events that were scheduled at most 20minutes before, “*NormalTakeOff*” is applied. Otherwise, “*LateTakeOff*” is applied

```

rule NormalTakeOff {
  when {
    toe: TakeOffEvent(sched+20 >= now);
    flight: toe.flight();
  } then {
    update flight.departed = true;
    update flight.on time = true;
  }
}
  
```

```

rule LateTakeOff {
  when {
    toe: TakeOffEvent(sched+20 < now);
    flight: toe.flight();
  } then {
    update flight.departed = true;
    update flight.on time = false;
    emit new FlightDelayEvent(
      flight.fl_no());
  }
}
  
```

Example Scenario : Airline Industry - Analytics

Business Rule for Analytics

Computes the number of late departures for every flight number

```
rule RAvG {  
  when {  
    late_count: aggregate {  
      f: Flight(on_time = false);  
      fn: f.fl_no();  
    }  
    groupby { fn }  
    do { count { f }; }  
  } then {  
    insert new FlightStat(  
      fn, late_count);  
    }  
}
```

Example1

- If a particular flight is frequently delayed, the system can trigger a review, or it can be taken into account when re-booking passengers

Example2

- If the system receives an event about a passenger missing their flight, the agent handling this event can consult available summary status information to look for alternative flights
- The system reports those alternatives back to the passenger, who can then re-book, leading to another event

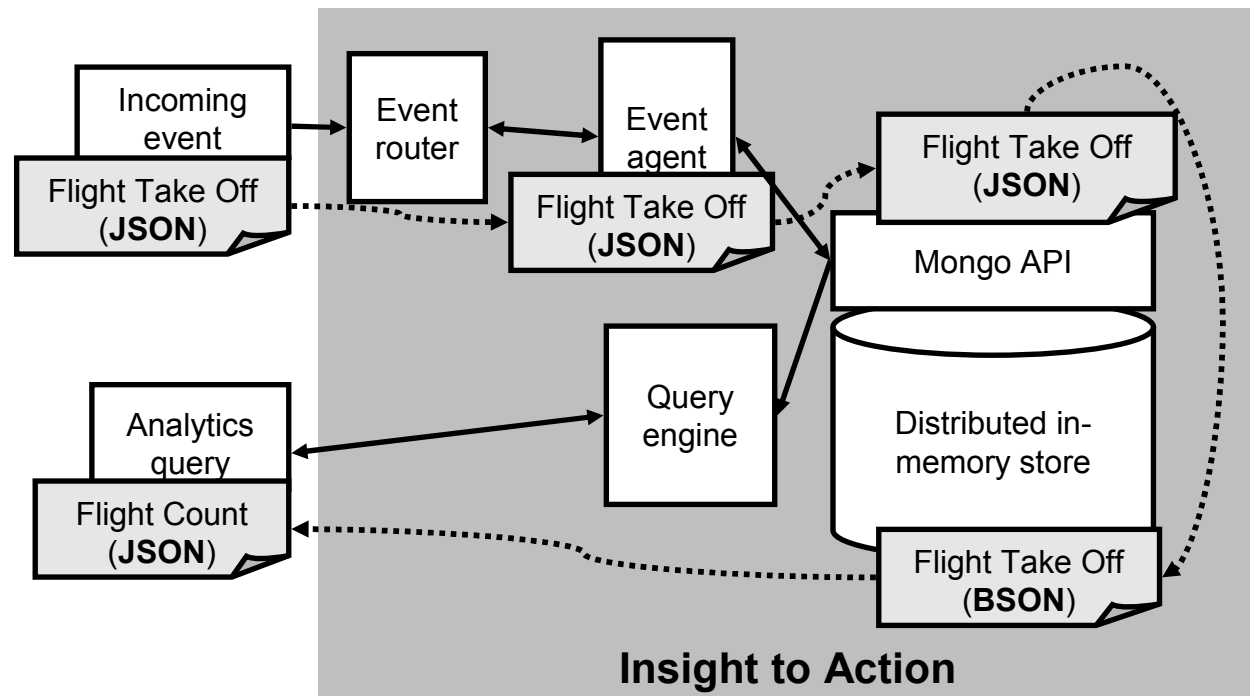
JSON support for Insight to Action

- One important aspect of our work is the built-in support for JSON as a data model throughout the system

JSON : JavaScript Object Notation (JSON) is consisting of attribute–value pairs and used for its easy integration with the client through JavaScript

Benefit for I2A

- Extend the data with new information
- Familiar with mobile devices
- Integrate with external services
– e.g. weather news



Integrated JSON store : JSON store for I2A



- MongoDB is one of the standard stores for JSON documents
 - Open-source document-oriented database for JSON
 - Use BSON (Binary JSON) to store JSON documents
 - Scales horizontally based on auto-sharding across multiple machines
 - Supports only atomic transactions on individual rows

Our challenge is..

How to reuse MongoDB's API without losing the tight integration with event agents?

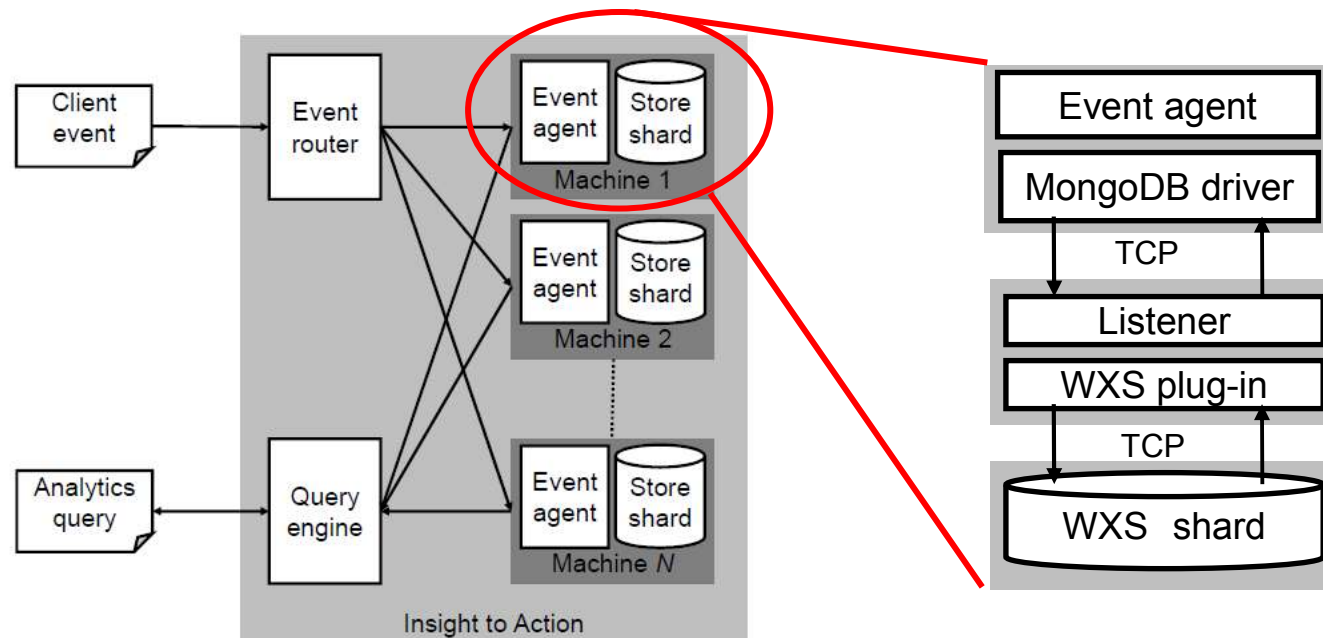


WebSphere eXtreme Scale (WXS)
Distributed in-memory store with MongoDB API

- In-memory Key/Value store
- Supports transactions
- Scalable with sharding

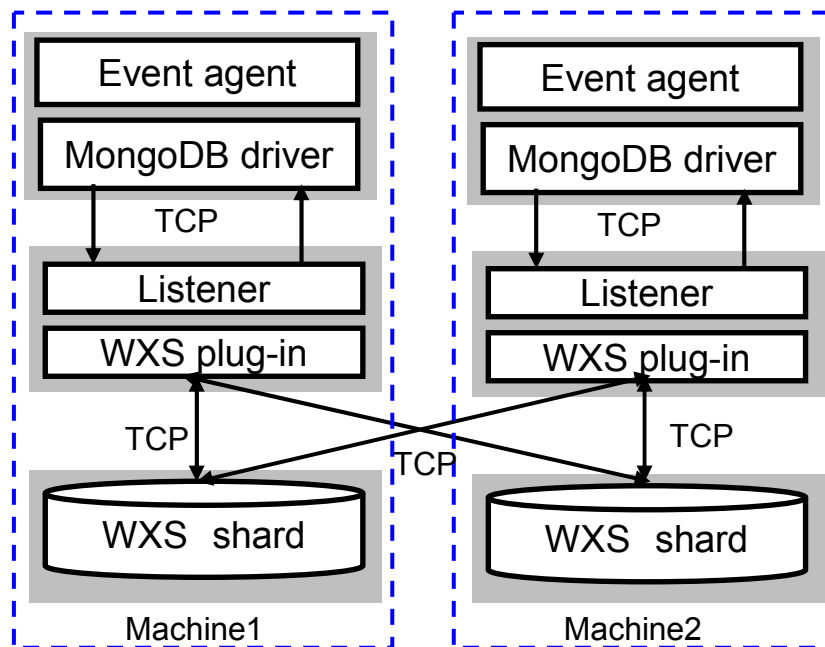
Integrated JSON store : JSON store for I2A

- The event agent communicates with the listener using the MongoDB driver and sends the query as serialized BSON data to the listener
- The listener is a server application on WXS that intercepts the MongoWire Protocol messages
- In the WXS shards, each query agent processes a query and then returns the requested data. The results are collected by the WXS plug-in, and serialized for transmission to the event agent

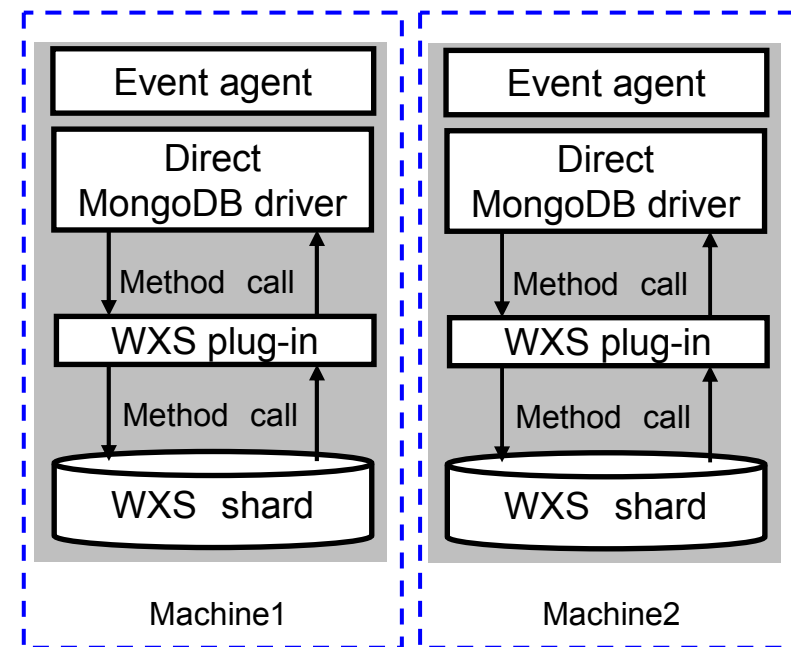
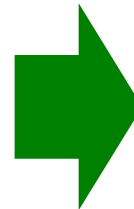


Performance Optimization : Embedded JSON store

- We developed the embedded store shown in (b) Embedded by eliminating the TCP communications from (a) External
 - Even without eliminating the TCP communications, performance is improved when the event agent accesses only the WXS shards on its own machine
 - This assumption is true thanks to the fundamental design of I2A: since the I2A architecture includes an event router, we do not need to fall back on WXS for routing to the proper shard



(a) External



(b) Embedded

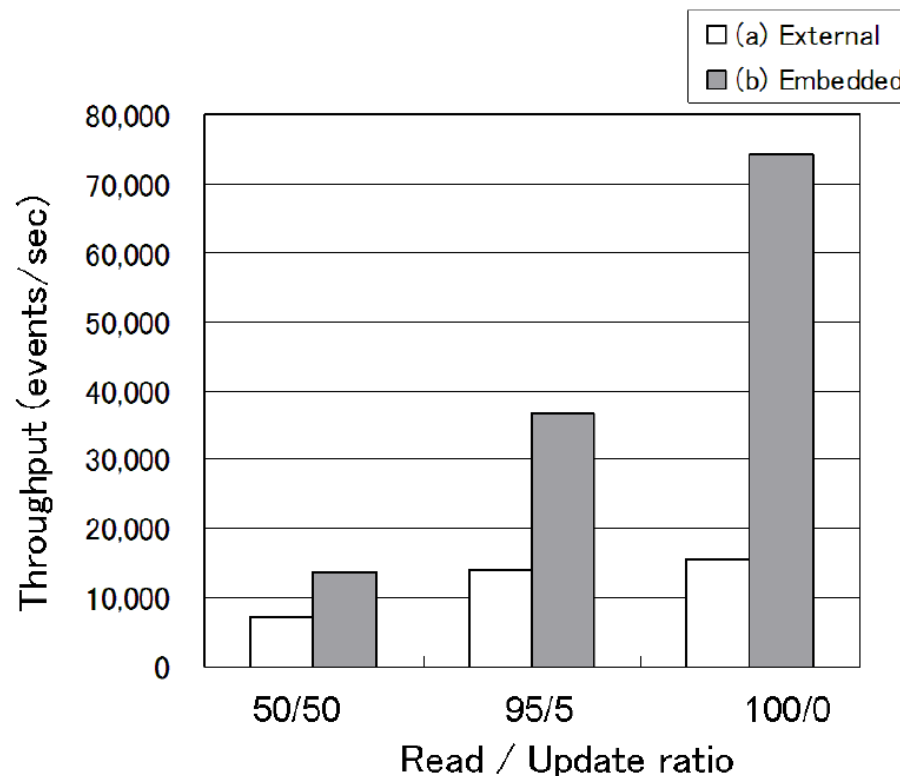
Experimental Evaluation

We evaluated the effectiveness of our embedded JSON store

- YCSB (Yahoo! Cloud Serving Benchmark) benchmark for emulation of event processing workload
 - Framework and common set of workloads for evaluating the performance of different "key-value" and "cloud" serving stores
 - Predefined drivers : MongoDB, Cassandra, HBase, Voldemode, etc.
 - Read/update ratio: 50/50, 95/5, 100/0
- Configuration
 - # of records = 100,000 ; # of operations = 5,000,000 ; Data size: 1 KB records (10 fields, 100 bytes each, plus key)
- Measurement
 - Throughput
 - Latency
 - Scalability

Throughput and Latency

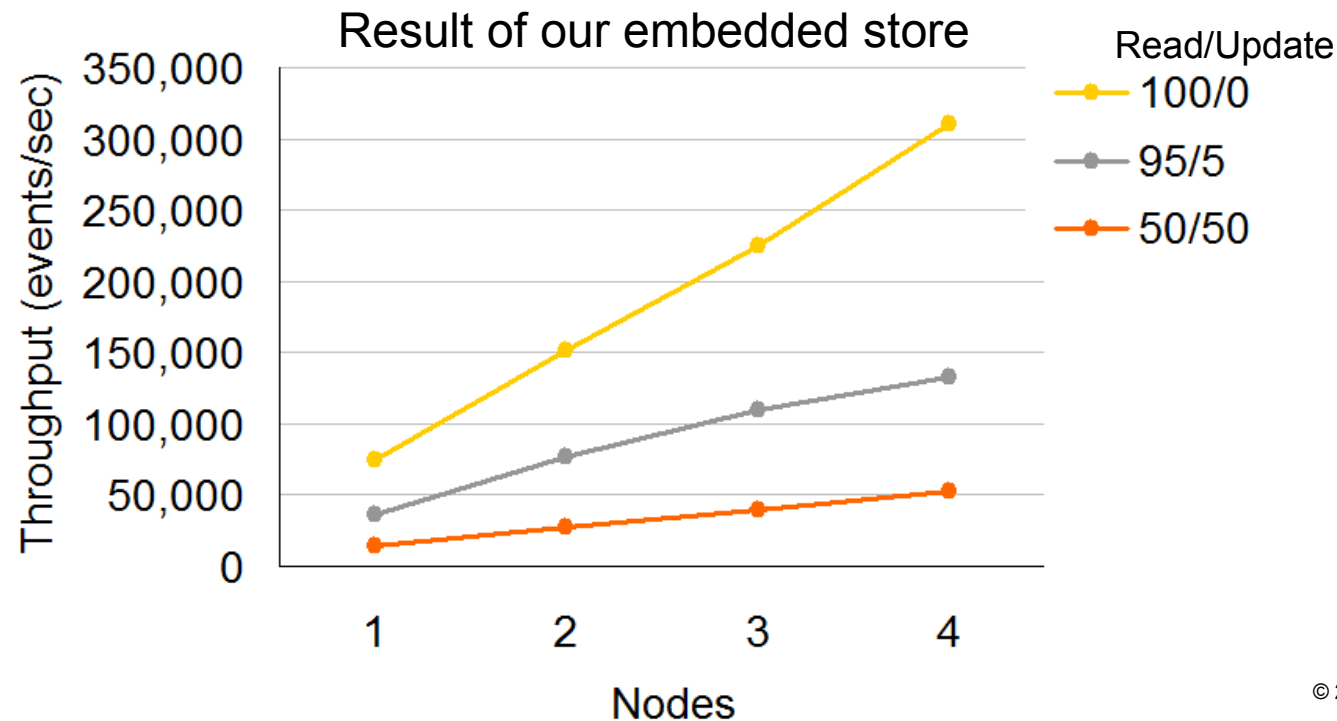
- The throughput of the embedded store was better for all read/update ratios
 - The result for the read-only scenario was about 4.8 times higher than the (a)External
- The embedded store reduced the average latency by 87%.
 - The 95-percentile embedded latency was also much smaller than the external latency
- This indicates that eliminating the TCP/IP communication is highly effective for I2A.



	External	Embedded
Average	640 μ s	48 μ s
Minimum	302 μ s	32 μ s
Maximum	66,639 μ s	40,731 μ s
95-percentile	623 μ s	66 μ s

Scalability

- In all of the workloads, the throughputs scaled well as the number of nodes increased
- This demonstrates the good scalability property of our proposed distributed JSON store for I2A



Conclusion and Future Work

- I2A with a JSON store enables simple, flexible, and scalable stateful event processing
 - The Insight to Action (I2A) system embeds event processing into a distributed in memory JSON store
 - We presented our architecture for reusing MongoDB APIs showed performance evaluation

- We are still actively developing I2A
 - Investigating several improvements, notably efficient execution strategies for aggregations
 - How to improve freshness for the analytics without interfering with transaction performance

Thank you for you attention!



Back up

