1. Motivation

- Internal states in real-world applications can expand to >100 GB.
- Cluster negotiators can frequently communicate with applications to preempt or grant resources.
- Real-world applications can experience periodic or abrupt workload variations.

Our goal is to support:
- transparent elasticity
- high availability
in latency-sensitive streaming system.

2. System Model

a. Programming model

```c
class Event{
    virtual INT64 GetHashCode();
    virtual void Serialize(STRING string);
    virtual void Deserialization(STRING string);
};
```

b. Execution model

```
call & ComsumeEvent(id, EVENT event);
this->ProduceEvent(id, EVENT event);
```

3. System Design

a. Functional primitives

```
Internal state management abstraction: detach application-level parallelism from OS-level concurrency.
```

b. Elasticity and availability

```
Chained backup
Dynamic thread rescheduling
```

4. Comparisons

<table>
<thead>
<tr>
<th>System</th>
<th>State management</th>
<th>State update</th>
<th>Horizontal elasticity</th>
<th>Vertical elasticity</th>
<th>Fault tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storm &amp; S4</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Stateless reconstruct</td>
<td>Stateless reconstruct</td>
<td>Recompute</td>
</tr>
<tr>
<td>Samza</td>
<td>External databases</td>
<td>Fine-grained</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Sync. checkpoint</td>
</tr>
<tr>
<td>D-Stream</td>
<td>RDD</td>
<td>Immutable</td>
<td>Not supported</td>
<td>Thread rescheduling</td>
<td>Recompute</td>
</tr>
<tr>
<td>SEEP &amp; SDG</td>
<td>User-aware</td>
<td>Fine-grained</td>
<td>State partition</td>
<td>State partition</td>
<td>Asyn. checkpoint</td>
</tr>
<tr>
<td>TimeStream</td>
<td>User-transparent</td>
<td>Fine-grained</td>
<td>Sub-DAG reconstruct</td>
<td>Sub-DAG reconstruct</td>
<td>Dependent recompute</td>
</tr>
<tr>
<td>ChronoStream</td>
<td>User-transparent</td>
<td>Fine-grained</td>
<td>Fine-grained reconstruct</td>
<td>Thread rescheduling</td>
<td>Asyn. delta checkpoint</td>
</tr>
</tbody>
</table>

5. Experiments

a. Does ChronoStream scale well?

b. Does ChronoStream support large-state applications?

c. Does ChronoStream support transparent horizontal scaling?

d. Does ChronoStream support transparent vertical scaling?

e. Does ChronoStream support efficient failure recovery?

6. On-going work: ChronoDB

- Extracting operation dependencies from user-defined programs.
- Optimizing system performance by leveraging application-level semantics.

Preliminary experiments on 5 popular benchmarks show that ChronoDB can scale near-linearly to 20 machines, yielding at least 10 times higher throughput than existing systems.

Acknowledgement

The authors would like to thank David Maier, Zhengping Qian, Zhou Zhao, and the anonymous reviewers for their insightful suggestions.