Serverless Event-Stream Processing over Virtual Actors

Philip A. Bernstein, Todd Porter, Rahul Potharaju, Alejandro Z. Tomsic[i],
Shivaram Venkataraman[i], Wentao Wu
Microsoft Corp.
{philbe, tporter, rapoth}@microsoft.com, alejandro.tomsic@lip6.fr,
shivaram@cs.wisc.edu, wentwu@microsoft.com

1. GOAL

A serverless event-stream processing service with pay-for-use and SLA's must be scalable, highly available, and low cost.

- Scalable – It must handle millions of streams with a high-variance ingestion rate and millions of continuous queries that are added and removed dynamically.
- Low cost – To use resources efficiently, it needs to optimize queries automatically, to grow and shrink elastically, to adapt automatically to changes in the workload, and to support billing for usage, not just for reserved capacity.
- High availability – It needs to detect and recover quickly from component failures. It must also be able to alter query plans of running queries without shutting them down.

Our goal here is to argue that the virtual actor model is an ideal platform on which to build such a stream processing system.

2. ACTORS & VIRTUAL ACTORS

Actors are single-threaded objects that do not share memory and communicate via asynchronous messages. They are lightweight in their memory and computational overhead. They are typically implemented by a programming framework (i.e., a runtime library) that executes at user-level. Among other things, this makes context-switching fast.

In traditional actor systems, an application explicitly creates an actor and places it on a named server. By contrast, in the virtual actor model [5], an actor is automatically activated on demand, as follows: An RPC is sent to an actor *K* based on *K*'s location-independent id. If *K* is not currently loaded, then the runtime picks a server *S*, runs *K*'s constructor at *S,* and invokes *K*. *K* remains active for future calls. After a period of inactivity, the runtime deactivates *K* to free up its resources.

The placement algorithm that picks *S* can optimize the choice, for example, to load balance across servers. The use of location-independent id's enables *K* to be reactivated on another server without disrupting actors that are communicating with *K*. This simplifies migration of actors to other servers. Activation-on-demand also simplifies actor recovery from server failures, since the actor will be re-activated the next time it is invoked.

3. WHY VIRTUAL ACTORS?

The following properties of an actor system make it an appealing platform for event-stream processing:

- Actors are lightweight – For efficiency, a system that runs a large number of queries needs the query execution context to be lightweight. Since actors are lightweight, one can assign each actor to run just one query or a sub-query of a larger query (i.e., one actor runs a stage or a set of operators).
- Actors do not share memory – This enables fine-grained control of queries. Each actor can be independently assigned to any processor. A control function can start, stop, or migrate a query without affecting other queries.
- Actors communicate via asynchronous messages. – This enables a query executing in one actor to pipeline a stream of messages to another query. It also enables sub-queries to execute in parallel, since each sub-query can send many messages in parallel. Parallel execution is needed for scalability across cores and servers.
- Each actor has a unique identity – This enables each query, to be controlled independently. It can be configured to send streams to or receive streams from particular queries.
- Stateful and stateless actors – Supports queries that do or do not maintain state. A stateless actor can be freely replicated, without synchronization, which enlarges the range of possible query execution plans (e.g., the degree of parallelism for a filter operator).
- Plug-in storage – An actor can use any storage system to save its state.
- Threads – Actors are single-threaded, which simplifies application programming.

The on-demand activation mechanism and automatic actor placement of the virtual actor model adds more benefits:

- Resource efficiency – a query is activated only when input data arrives, and its resources are released after a period of inactivity. This on-demand usage model simplifies usage-based cost accounting.
- Load Balancing – queries can be load-balanced across servers by the actor placement algorithm that assigns queries to servers. On-demand activation simplifies the migration of running queries to other servers to balance the load.
- Recovery – every query, *Q*, that was running on a failed server will be reactivated automatically by a stream or query that sends an event to *Q*.
- Elasticity – if a lightly-loaded server is deprovisioned or a new server is added to a cluster, on-demand activation and automatic actor placement will dynamically adjust the system to the new configuration without additional mechanisms.

4. OPPORTUNITES & CHALLENGES

Although the virtual actor model automates the re-activation of queries that were running on a failed server, it leaves open the well-known challenge of recovering each query's state. The storage plug-in should be chosen to minimize recovery time [2].

The ability to handle a large number of *ad hoc* continuous queries enables interactive monitoring and debugging of IoT and telemetry systems. However, it requires automated query optimization, including sharing of subqueries [3] and in-place re-optimization of running queries. The latter requires plan migration [4], which should be non-disruptive.

An actor-oriented database can enable indexed access and *ad hoc* queries over intermediate output of continuous queries running as actors [1], for example, to enable inspection of partial results of queries over large windows.

5. REFERENCES

[1] Philip A. Bernstein, Mohammad Dashti, Tim Kiefer, David Maier: Indexing in an Actor-Oriented Database. CIDR 2017

[2] Jeong-Hyon Hwang, Magdalena Balazinska, Alex Rasin, Ugur Çetintemel, Michael Stonebraker, Stanley B. Zdonik: High-Availability Algorithms for Distributed Stream Processing. ICDE 2005: 779-790

[3] Alekh Jindal, Konstantinos Karanasos, Sriram Rao, Hiren Patel: Selecting Subexpressions to Materialize at Datacenter Scale. PVLDB 11(7): 800-812 (2018)

[4] Luo Mai, Kai Zeng, Rahul Potharaju, Le Xu, Steve Suh, Shivaram Venkataraman, Paolo Costa, Terry Kim, Saravanan Muthukrishnan, Vamsi Kuppa, Sudheer Dhulipalla, Sriram Rao. Chi: A Scalable and Programmable Control Plane for Distributed Stream Processing Systems. *PVLDB 11(10)*: 1303-1316 (2018)

[5] Orleans, http://dotnet.github.io/orleans.

[i] Work done at Microsoft Research