Welcome to This Week on developerWorks. I'm your host, Calvin Powers. You know, there's more and more stream processing applications coming online every day. The Internet scale and real-time processing platforms like InfoSphere Streams, it's extremely useful in many different application domains.

But the real-time results in InfoSphere Streams can provide a lot of values only if you can share it with other applications and other people also in real time.

My guests this week have a solution to that problem which they have just published on developerWorks in an article called Using InfoSphere Streams with memcached and Redis. Our first guest is Dr. Bugra Gedik. He is an assistant professor at Bilkent University in Ankara, Turkey. Bugra, welcome to This Week on developerWorks. Tell us a little bit about your work at the Bilkent and with IBM.

GEDIK: Thank you for having me. I do research in data-intensive distributed systems, with a particular focus on fast data and big data. I actually worked at IBM Watson Research Center for six years before joining Bilkent University.

And at IBM I was a member of the team that built IBM InfoSphere Streams, and I'm still continuing to collaborate with my IBM colleagues on streams-related research and technology such as access to shared state, which is something we will briefly talk about today, and other things like auto parallelization for stream processing.

POWERS: We also have with us a longtime developerWorks contributor, Mr. Senthil Nathan. He's a senior technical staff member at IBM's T.J. Watson Research Center. Senthil, welcome back to This Week on developerWorks. Tell us a little bit about what you do up there at Watson Research Center.

NATHAN: Thank you for inviting us. I've been at Watson Research for about 21 years, and for the past seven years I have been part of a team that has been working on stream processing.

As you may already know, Streams as a product had its origins inside IBM Watson Research in the name of System S. My particular role there is take the wonderful components inside Streams, SPL programming model and Streams runtime, take those two components to customers and gain adoption. So, I work on proof of concepts. I publish tons of examples on developerWorks. Sometimes I go to customer places and work with their engineers in building streaming solutions.

POWERS: Senthil, can you describe the challenge that you and Bugra address in this article?

NATHAN: Streams has traction in many industries, most importantly public sector, financial sector, telco, health care. In those areas, streaming applications by its very nature is heavily distributed. So, you will string together several streams operators that deploy that application either on multicore machine, on a single machine, or a cluster of machines.

So, in its current state, in Streams, we cannot share local state across operators. So, that is a gap we have today. So, in this technology we built called Distributed Process Store, Bugra and I, we tried to address that gap. We tried to provide a solution that you can share state across multiple operators running on single machine or multiple machines.

POWERS: And Bugra, can you elaborate on that, the motivation any, and then sort of outline the components in your solution?

GEDIK: In traditional stream processing, applications only have access to local state, as Senthil has mentioned. And that's how things are in SPL today without this DPS toolkit.

And normally data flows between operators only via stream connections. However, there are many applications where shared state can be accessed across operators' processing elements and machines is an important use case.

One example is to manage dynamic configuration data that's needed by multiple operators in a stream processing application. Another use case is to access large-scale data, large-scale state that actually doesn't fit into the memory of that single machine. So, you really need access to multiple machines. You need to store your state in multiple machines.

Yet, another example use case is to provide some external system access to the streaming application data. So, you have your SPL application, but you also have some external systems that may want to access the data that the SPL application
processes.

So, in this work, we provide support for distributed shared state in SPL to address these use cases. The core idea or approach is to provide a key-value store API via the use of SPL native functions.

So, using these APIs, users -- SPL users can create stores of key-value pairs in their Streams code, such as written custom operators in SPL. Using these APIs, they can perform basic operations such as put, get, remove, and iteration over the stores, over the data stores. And interestingly all SPL types can be used both as keys and values. So, it's a very flexible API.

Additionally, we provide locking support. So, if you want to perform multiple operations as an atomic unit within a mutual exclusion block, you could also do that.

Now, how does this work? So, behind the scenes the state is stored in a back-end system. Right now we support two back-end systems. These are memcached and Redis. Both of those systems are open source...open source SoftLayer. We use them as distributed key-value stores.

So, what we do is that we, by using these systems we distribute the data items over multiple machines and then serve, put, and get requests from the stream processing operators that may reside in many different places, like from different machines, in a scalable manner.

So, memcached is a lean implementation that does not have persistence or replication support. So, it's very lightweight, but it's not feature rich. Redis is another alternative that you could use as a back end, and it's more feature rich. It supports replication and persistence.

Interestingly, if you want to extend the DBS toolkit for additional back ends, we provide a very easy interface. So, by just extending the interface that we provide, you could actually add new back ends to the system. If you have your own favorite distributed key-value back end, you may be -- you could add that into the DPS toolkit with little effort.

POWERS: By exploiting these APIs that you provide, it gives the stream processing application the ability to sort of publish almost a real-time dashboard type of system for other applications to access. Is that right?

GEDIK: That's one of the use cases that we discussed, giving access to stream processing application state to external applications. That would be one use case. Another important use case is to actually share that state within different operators inside the same stream processing application.

POWERS: Gotcha. Okay. And I know that you have extended the programming APIs into C++ and Java, and I know that performance is extremely important in these types of applications. Senthil, I was wondering if you would tell us a little bit about what you folks did to ensure that there was a good performance in this solution.

NATHAN: When we picked a back-end store, we already surveyed a few options, and we picked the top two that is widely used today. Redis is highly performant and memcached has been around for about 10 years. It's highly performant.

But it's, we want to highlight one important thing to the audience. You have to remember, we are going to an external server, remote server. So, that means from your Streams operators, we are going to additional server.

That means that you have to pay the cost of network transport here. So, they're not going to be super-duper fast, but still it is acceptable level of performance. As Bugra already mentioned, you can now go, since DPS is so highly extensible, you can bring your own high-performance distributed cache systems that you have built, and you can bring and attach it to our system.

For example, there is ongoing research work in Watson, which is called RCache, which is supposed to...which is expected to work on RDMA and RoCE interfaces, which is likely it will perform much better than Redis and memcached. So, that's futuristic work we are looking at. Maybe next year when RCache is ready we'll connect it with DPS. So, that is our performance answer.

POWERS: I look forward to having you folks back on next year to talk about that RCache implementation. So, don't forget to write the developerWorks article.
That's going to be all the time we have for this episode of developerWorks. I want to thank Senthil and Bugra for sharing their article with us, and I know this is going to be interesting to anybody who has an interest in writing performant InfoSphere Streams applications. Don't forget we'll have a link to Senthil and Bugra's article on ibm.com/developerworks/thisweek.

[ MUSIC ] [END OF SEGMENT]