Pravega Release 0.5.0 Announcement

Table of Contents

Pravega summary

Pravega is storage for streams. It is a software system that enables applications to write streams of events or bytes in an append-only manner. It offers a number of desirable features for stream applications, including the ability to auto-scale streams, write transactionally, and even build replicated state machines via a state synchronizer.

Pravega has three core components:

- **Controller**: manages the lifecycle of streams and transactions
- **Segment store**: implements the functionality in the critical path of reads and writes

- **Client**: library for writer applications that append data to Pravega streams and reader applications that access the data

To deploy Pravega, we have focused recently on the development of a Kubernetes operator, and that is the primary way we recommend to deploy a Pravega cluster. Other methods available to run Pravega are: manual installation, standalone (for testing and development), and Docker Swarm.

To enable applications to process data in and out of Pravega, we have developed, along with the Apache Flink community, a connector to be able to build stream pipelines on Pravega and Flink.

Pravega is an open-source system released under the Apache 2.0 License. The source code is available in a GitHub repository: http://github.com/pravega/pravega

The Pravega Operator for Kubernetes follows an independent release cycle. The source code is available in this GitHub repository: http://github.com/pravega/pravega-operator

The Flink connector follows the same release cycle as Pravega, and it is maintained in a separate GitHub repository: https://github.com/pravega/flink-connectors

What's new?

Here we cover the highlight changes in this release across the main components of Pravega.

In Pravega

The main theme of Pravega 0.5.0 is scalability. We have made deep changes to make the system more scalable, specifically:

- **Stream metadata**: we have implemented a new table API in the segment store that enables us to store stream metadata in Pravega itself, removing the memory limitation we previously had with ZooKeeper servers. With this change, we can accommodate a much larger number of streams and even more metadata for streams individually.
- **Connection pooling**: we have added the ability of sharing connections between components of Pravega to avoid an explosion of the number of connections. This change is important, for example, to be able to accommodate a larger number of clients connecting to a Pravega cluster.

In this section, we list and detail the highlights of the release. For a complete list of the changes, see the release description on GitHub:

https://github.com/pravega/pravega/releases/tag/v0.5.0

Table segments

One of the many uses of an append-only data structure like a segment is to store the changes to a table of key-value pairs. Calls to put or delete are recorded in a segment, and it is possible to materialize such a table by reading the changes in sequence and applying them.

This principle is the basic one we have used to develop table segments. The segment store from this version exposes an internal API that enables the controller to create tables backed by Pravega segments, and execute requests to put, get, and remove key-value pairs. This feature is relevant to enable the controller to store internal metadata in a scalable fashion, leveraging the scalability properties of the segment store.

Each table segment has an associated index that enables key lookups. The index is based on a hash table using extended attributes. To support such an extensive use of extended attributes beyond the cases we were already using it for (*e.g.*, writer IDs), we replace the way we organize such attributes. Up to branch 0.4, we were storing updates linearly and compacting the data into snapshots regularly. This scheme works well for a relatively small number of attributes. To enable fast lookups with a larger number of attributes, we implemented a B+ Tree.

Stream metadata

Pravega streams comprise one or more segments, and the set of segments in a stream can evolve over time according to workload. Consequently, to be scalable, Pravega must be able to:

- Accommodate changes to a single stream over the lifetime of the stream. The amount of metadata per stream can grow without bounds, depending on the frequency of scaling and number of parallel segments.
- Accommodate a large number of such streams

The first point has been covered in the 0.4 release branch, and is present in all 0.4.x releases. For a description of how we addressed the metadata scalability issue per stream, see PDP-32:

https://github.com/pravega/pravega/wiki/PDP-32:-Improve-scalability-of-Controller-Metadata

We address the second point in Pravega 0.5.0. In this release, we store stream metadata in table segments rather than in Apache ZooKeeper, which is what we have been doing up to branch 0.4. ZooKeeper is not designed to store an unbounded amount of data in its in-memory database, and the capacity is limited by the memory of a single server. We continue to use ZooKeeper for general coordination, including serving ledger metadata for Apache BookKeeper.

By default, the stream metadata is stored using table segments. See PDP-35 for more detail of the design:

https://github.com/pravega/pravega/wiki/PDP-35:-Move-controller-metadata-to-KVS

Connection pooling

Connection pooling is the ability to share a network connection between two endpoints. For example, a client typically appends to multiple segments and some of these segments might be hosted by the same segment store instance. The client can use the same network connection to send append requests to the segment store. Similarly, a controller instance can use the same connection when sending requests to a segment store instance.

The feature implemented in this version enables this kind of sharing, it reduces the overall number of connections needed to operate, making the system more scalable. It is important to note, however, that connection pooling has introduced a performance penalty to the append path. This penalty is related to the way we perform batching between the client and the segment store. We will look into fixing this issue going forward. In the meantime, there is a flag introduced in this pull request to disable pooling on the append path:

https://github.com/pravega/pravega/pull/3952

See the PDP-36 design document for more detail about the implementation:

https://github.com/pravega/pravega/wiki/PDP-36:-Connection-Pooling

Security

There are no major changes related to security in this release. We have improved the following areas:

Authorization model

The authorization model in Pravega 0.4 allowed for some undesirable behavior, in particular with respect to operations on scopes. See the pull request and associated issue for more detail about the specific problems and the implemented changes to address them:

https://github.com/pravega/pravega/pull/3386

TLS Configuration for distributed deployments

We have added missing configuration parameters to:

- enable the controller REST server to expose an HTTPS endpoint;
- enable TLS between Zookeeper client and server.

See the pull request for a discussion of the parameters:

https://github.com/pravega/pravega/pull/3586

Metrics

We have added support for Micrometer metrics reporting. Micrometer provides key features that support other improvements in this release. One of the key features is dimensionality: the ability of enriching metric names with key-value pairs.

We have removed Coda Hale (Dropwizard) metrics support in favor of Micrometer.

https://micrometer.io/

Tests

We have tested Pravega extensively during the preparation of release 0.5.0. Here are a few concrete points about our testing:

- We have currently over 2,000 Junit test cases that we exercise on a regular basis.
- We have currently 30 system test cases that we exercise on a regular basis. Our system tests currently run on both Kubernetes and Docker Swarm.
- We have executed longevity workloads on different infrastructures: Google Cloud, Amazon Web Services, and on-premises PKS.
- We have performed a number functional and non-functional tests, such as killing Kubernetes pods and observing the system recover.

JDK 9+

Over the course of our testing, we came across a bug in JDK that caused readers to stall and consequently reader groups due to checkpoints not completing. See Pravega issue 3900 for more detail including references to the OpenJDK reports we have found:

https://github.com/pravega/pravega/issues/3900

The bug is not resolved in JDK 8. Consequently, to avoid the reader stalls, we recommend running Pravega applications on JDK 9+. We have not encountered similar issues with the other components of Pravega, but we still recommend using later versions of the JDK when possible. The issue is related to complex completable future chains, and we make extensive use of completable future across Pravega.

In the Pravega Kubernetes Operator

The first release of the operator, v0.1.0, is from the end of September. Since then, the operator has evolved in various ways, including its robustness. The latest release of the operator is v0.4.1, which is a bug fix release for v0.4.0. Version v0.4.0 includes:

- The ability to update a Pravega cluster to a newer version
- A new admission webhook that validates requests to Pravega clusters
- Client-Server TLS configuration support
- A new Helm chart to install a Pravega cluster
- Support for OpenID Connect (OIDC)

In the Flink Connector

The v0.5.0 version of the connector:

- Updates Flink version from 1.6.0 to 1.7.2
- Adds Flink TableFactory support for Pravega
- Enables Flink SQL client access support for Pravega

Published artifacts

For Pravega:

GitHub:
https://github.com/pravega/pravega/releases/tag/v0.5.0

Docker image:
https://hub.docker.com/r/pravega/pravega

Maven:
https://search.maven.org/search?q=g:io.pravega

For the Pravega Kubernetes Operator:

GitHub:
https://github.com/pravega/pravega-operator/releases/tag/v0.4.1

Docker image:

https://hub.docker.com/r/pravega/pravega-operator/tags

For the Flink Connector

GitHub:
https://github.com/pravega/flink-connectors/releases/tag/v0.5.0

Maven:
https://search.maven.org/search?q=a:pravega-connectors-flink_2.12