Seamless State Management for Distributed Stream Processing Framework

Pritish Mishra

Progress Report

Project Website: https://pritishmishra.github.io/projects/uoft

1 Achievements

1. State management system framework is built to support basic functionalities for stateful applications. The framework encapsulates and abstracts common datatypes used as in-memory state objects such as Hashmap and provides the user a modified version of these data-types. The user can perform all activities similar to the original data-type variable. However, in this case, the framework can track the state changes made to the in-memory data-variable and manage the state backup and restore functionalities based on this. This approach makes the process of state management agnostic from the application developer and puts it entirely in the purview of the state management framework of stream processing systems. The application developer can solely focus on developing the logic of applications, with the only additional task of using data-types provided by the framework instead of standard ones provided by the languages.

2. A basic partitioning strategy is developed to support indexing of the state. Consistent hashing technique is used to divide the state into unique indexes and the hashing algorithm determines the location of state partition. The unique state partitions are accessible by their unique state representations called partition keys. No two partition keys are same and no two replicas of an operator handles a single partition key. In other words, only one operator replica is responsible for the particular partition of the state denoted by the partition key. Currently, the framework isn't built to handle load-balancing concerns. So, all the partition keys are assigned to single replica. Upon reconfiguration, only one partition key is supposed to be moved from one replica to the other. This is just to ensure seamless reconfiguration is possible and there is no other objective of moving the partition key from one location to another.

3. No specific implementation has been done for state storage system. The requirement for an ideal state storage system is it has to be geo-distributed to support the edge based scenarios of the framework. Additionally, it has to be used to take backup of the data on one node in the network hierarchy (cloud) and the restore of the data could be performed at another node in the network hierarchy (edge). Initial expts were done using RocksDB and a simple backup agent was used to perform this task. But, since stock version of RocksDB isn't geo-distributed, there was a need to develop a mechanism to handle node-to-node data migration. Instead of implementing all these features by hand, a better approach was to use some project which already provides such features out-of-the-box. One such project is PathStore (or Feather [2]) which is a geo-distributed database that consolidates the data replication at each node of the hierarchy. However, there were some features missing in this system, since it wasn't originally developed to handle use-cases such as ours. For example, the restore mechanism of Pathstore is reactive rather than pro-active. It fetches the data from a parent node, if and when it is requested by a client at the child node. For our case, however, we would need the restore to proactive and all data must be moved immediately during the migration. So, a warmup service had to be added to implement this feature.

4. Lastly, a mechanism for orchestrating all these features has been developed. First, the state management system accompanies all operators and takes backup of the state variables. It performs two levels of backup - periodic and on-demand. The periodic backup occurs asynchronously and is triggered as per a configured frequency. However, the on-demand backup is synchronous and stops the execution of the tuples. It is triggered only when a reconfiguration occurs and state migration needs to happen. Second, there is a centralised reconfiguration system that manages the spawning of operators in new locations and performs appropriate routing. This reconfiguration system provides triggers or signals to the state management system running alongside each operator. One signal indicates when reconfiguration is started and hence, on-demand backup needs to be triggered and the other signal is when the restore is complete and the normal processing of tuples can resume.

2 Pending Tasks

1. Evaluation of the framework is the major task remaining. First an application needs to be determined that can be used to reliably emulate the common edge scenarios. This application should also have provision for stateful operators. It should contain state partitioned into unique indexes too.
2. Then, the scenarios for measuring the performance of the system have to be developed. These scenarios should showcase how seamless reconfiguration benefits various metrics of the system like application end-to-end-latency, throughput, etc.
3. A comparison needs to be done with state-of-the-art solutions that provide similar functionalities. Else this could also be done with the proposed system having the functionality of reconfiguration versus without it.
4. Depending upon the evaluation results, optimisation tasks may come up to tune the system appropriately.

3 Blockers

No blockers as such as of now.

4 Conclusion

Overall, the system design and implementation seem to be complete. Evaluation of the system is the major task remaining. During the evaluation, it is expected that the system might have to be further optimised or certain components have to be redesigned.

References

[1] Mortazavi, Seyed Hossein, Mohammad Salehe, Moshe Gabel, and Eyal de Lara. "Feather: Hierarchical querying for the edge." In Proceedings of the Fifth ACM/IEEE Symposium on Edge Computing (SEC). IEEE. 2020.