Stocator: An Object Store Aware Connector for Apache Spark[*]

Gil Vernik, Michael Factor, Elliot K. Kolodner, Effi Ofer
IBM Research – Haifa
(gilv,factor,kolodner,effio)@il.ibm.com

Pietro Michiardi, Francesco Pace
Eurecom
(pietro.michiardi,francesco.pace)@eurecom.fr

ABSTRACT

Data is the natural resource of the 21st century. It is being produced at dizzying rates, e.g., for genomics, for media and entertainment, and for Internet of Things. Object storage systems such as Amazon S3, Azure Blob storage, and IBM Cloud Object Storage, are highly scalable distributed storage systems that offer high capacity, cost effective storage. But it is not enough just to store data; we also need to derive value from it. Apache Spark is the leading big data analytics processing engine combining MapReduce, SQL, streaming, and complex analytics. We present Stocator, a high performance storage connector, enabling Spark to work directly on data stored in object storage systems, while providing the same correctness guarantees as Hadoop's original storage system, HDFS.

Current object storage connectors from the Hadoop community, e.g., for the S3 and Swift APIs, do not deal well with eventual consistency, which can lead to failure. These connectors assume file system semantics, which is natural given that their model of operation is based on interaction with HDFS. In particular, Spark and Hadoop achieve fault tolerance and enable speculative execution by creating temporary files, listing directories to identify these files, and then renaming them. This paradigm avoids interference between tasks doing the same work and thus writing output with the same name. However, with eventually consistent object storage, a container listing may not yet include a recently created object, and thus an object may not be renamed, leading to incomplete or incorrect results.

Solutions such as EMRFS [1] from Amazon, S3mper [4] from Netflix, and S3Guard [2], attempt to overcome eventual consistency by requiring additional strongly consistent data storage. These solutions require multiple storage systems, are costly, and can introduce issues of consistency between the stores.

Current object storage connectors from the Hadoop community are also notorious for their poor performance for write workloads. This, too, stems from their use of the rename operation, which is not a native object storage operation; not only is it not atomic, but it must be implemented using a costly copy operation, followed by delete. Others have tried to improve the performance of object storage connectors by eliminating rename, e.g., the DirectParquetOutputCommitter [5] for S3a introduced by Databricks, but have failed to preserve fault tolerance and speculation.

Stocator takes advantage of object storage semantics to achieve both high performance and fault tolerance. It eliminates the rename paradigm by writing each output object to its final name. The name includes both the part number and the attempt number, so that multiple attempts to write the same part use different objects. Stocator proposes to extend an already existing success indicator object written at the end of a Spark job, to include a manifest with the names of all the objects that compose the final output; this ensures that a subsequent job will correctly read the output, without resorting to a list operation whose results may not be consistent. By leveraging the inherent atomicity of object creation and using a manifest we obtain fault tolerance and enable speculative execution; by avoiding the rename paradigm we greatly decrease the complexity of the connector and the number of operations on the object storage.

We have implemented our connector and shared it in open source [3]. We have compared its performance with the S3a and Hadoop Swift connectors over a range of workloads and found that it executes many fewer operations on the object storage, in some cases as few as one thirtieth. Since the price for an object storage service typically includes charges based on the number of operations executed, this reduction in operations lowers the costs for clients in addition to reducing the load on client software. It also reduces costs and load for the object storage provider since it can serve more clients with the same amount of processing power. Stocator also substantially increases performance for Spark workloads running over object storage, especially for write intensive workloads, where it is as much as 18 times faster.

CCS CONCEPTS

• **Information systems → MapReduce-based systems**; **Cloud based storage**;

REFERENCES

[1] Jeff Barr. 2014. Amazon EMRFS Blog. (2014). https://aws.amazon.com/blogs/aws/emr-consistent-file-system/
[2] Chris Nauroth. 2016. Apache Hadoop S3Guard JIRA. (2016). https://issues.apache.org/jira/browse/HADOOP-13345
[3] Gil Vernik. 2017. IBM Stocator Source Code. (2017). https://github.com/SparkTC/stocator
[4] Daniel C. Weeks. 2014. Netflix S3mper Blog. (2014). http://techblog.netflix.com/2014/01/s3mper-consistency-in-cloud.html
[5] Reynold Xin. 2016. [SPARK-10063][SQL] Remove DirectParquetOutputCommitter. (2016). https://github.com/apache/spark/pull/12229

[*]The research leading to these results has received funding from the European Union Horizon 2020 Research and Innovation Programme (grant agreement 644182 – IOStack)

SoCC '17, September 24–27, 2017, Santa Clara, CA, USA
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5028-0/17/09.
https://doi.org/10.1145/3127479.3132569