Elastic Scaling in Actian Avalanche

Steffen Kläbe
steffen.klaebe@actian.com

October 2, 2019

Actian Avalanche is a cloud data warehouse solution designed to run analytical workloads on large-scale datasets and runs in hybrid cloud environments, combining on-premise environments and public clouds. Moving their applications to the cloud, users benefit from various advantages of the cloud environment, e.g., the availability of a nearly inexhaustible amount of resources accessible from everywhere, a simple and predictable pay-per-use pricing model and the elastic scaling of resources to fit varying demands. In order to exploit the flexible hardware environment beneath the data warehouse system, we introduce the elastic scaling feature for Actian Avalanche.

Scalability is a core characteristic of massively parallel processing (MPP) systems. As a static property, it defines how a system behaves for a specific configuration. Related to a distributed data warehouse system, scalability describes the query runtime as a function of the assigned computing nodes. Ideally, query runtime decreases when adding more computing resources.

On the contrary, elasticity is a dynamic property. It is defined as the degree a system is able to adapt to workload changes by provisioning and deprovisioning resources in an automatic manner. Elasticity aims at matching provided resources with the current system load at every point in time.

An Actian Avalanche installation can be scaled using a system restart and a reconfiguration while the system is stopped. This has several drawbacks. First, the system replays transaction logs during the starting phase, which is a possibly long-running operation depending on the log size. Furthermore, all system buffers are empty after a restart, leading to storage access and therefore slowing down the first queries after scaling. Another major drawback is the system's unavailablity during the scaling process, which might be critical for business applications.

To overcome the drawbacks of this "inelastic" way of scaling, we introduce the elastic scaling feature for Actian Avalanche. We first discuss the workflow of the scaling process before explaining how work is distributed in the scaled environment. As an optional optimization we present the buffer matching mechanism designed to provide the full system performance after a scaling process.

The new scaling workflow

Scaling Actian Avalanche can be explicitly triggered by the user providing the desired cluster size. The internal process of scaling up a running Actian Avalanche cluster is basically divided into three steps. First, additional nodes are acquired from the cloud service provider like shown in Figure 1. In the second step, Avalanche is started on the new nodes and they are added to the existing cluster by synchronization with the already running Avalanche nodes. To make the added nodes able to participate in query execution, data responsibilities are moved from existing nodes in a third step.

Downscaling the system follows the reverse procedure. First, data responsibilities are moved away from Avalanche nodes that are planned to shut down. These nodes are chosen by the system itself after the user decided for downscaling. In the second step, the Avalanche cluster is split into the group of remaining nodes and the group of removed nodes. While the remaining nodes synchronize themselves and are immediately ready to continue with the execution of incoming queries, the group of removed nodes is shut down in the third step.

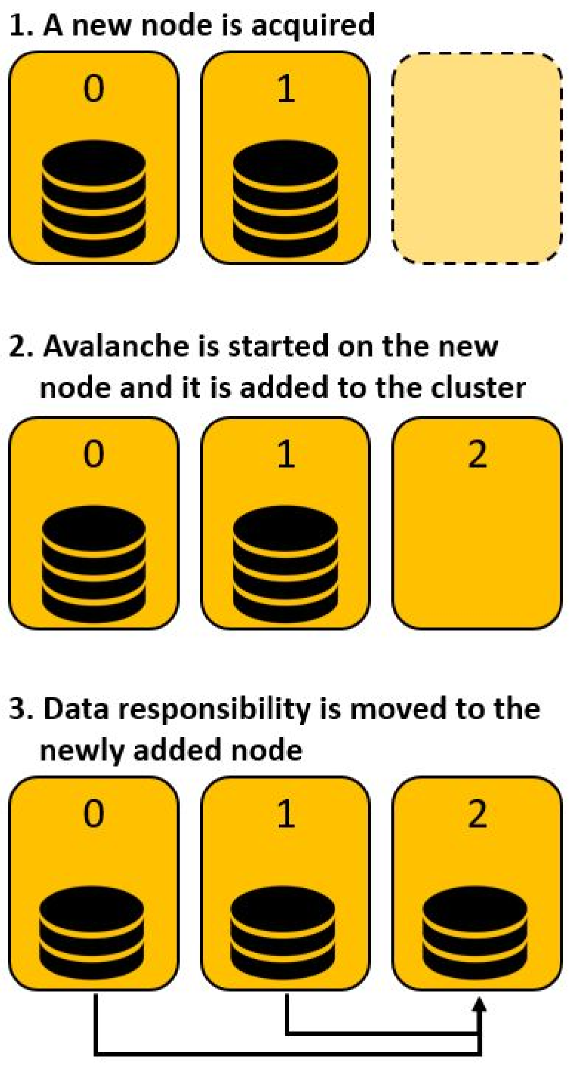
1. A new node is acquired
0
1
2. Avalanche is started on the new node and it is added to the cluster
0
1
2
3. Data responsibility is moved to the newly added node
0
1
2

Figure 1: *Scaling process for adding nodes*

The scaling process does currently not support concurrent user sessions in the system, which especially makes it unable to serve read or update transactions during the scaling process. Therefore, all user sessions have to be finished before starting the scaling process.

Work distribution

Actian Avalanche uses partitioning as the key concept to distribute work among system nodes. Partitioning is defined as an assignment of each tuple to a set $R_i, i \in \{0, \ldots, d-1\}$ of tuples, with R_i called a partition. This assignment is based on a partitioning function f applied to a partitioning key k, mapping the domain of k to the set of partitions $\{0, \ldots, d-1\}$, while k is also possible to be a combined key. A tuple t is assigned to partition R_i with $i = f(k(t))$. These partitions are afterwards assigned to nodes who hold the responsibility to execute a query on this partition, which is the key to distributed query processing. This partition assignment is stable for consecutive queries to enable data buffering.

In order to make the partition assignment elastic, Actian Avalanche uses the approach of overpartitioning. Running on a cluster of n nodes, the initial number of partitions d for a table is chosen to be reasonably higher than the number of nodes n. This way, each node becomes responsible for several partitions while the system is able to scale to d nodes without the need to repartition data to keep all nodes busy.

The Actian Avalanche partition manager realizes the partition assignment to nodes while also satisfying requirements of the elastic environment:

1. **Load balancing:** Assigning an equal number of partitions to each node is crucial to achieve an optimal query performance. As partitions are already created during table creation, the assignment strategy can only affect the number of partitions per node, not the size of each partition or their total number respectively.
2. **Lookup time:** The mapping $partition \mapsto node$ is evaluated numerous times within each query. Therefore, the partition manager provides constant lookup time for the mapping.
3. **Update time:** Scaling the cluster leads to changes in the partition assignment to achieve a balanced assignment. The partition manager provides linear update time for the partition assignments.
4. **Keeping co-locality of foreign-key related tables:** The key for node-local processing of join operators is the co-locality of partitions containing all join partners. The partition manager ensures the co-locality of these partitions even after scaling processes.
5. **Minimizing the number of partition reassignments:** A node that becomes responsible for a partition has no buffered data for this partition, leading to storage access for queries after the reassignment. Therefore, the partition manager minimizes the number of reassigned partitions that is necessary to achieve a balanced assignment.

As its main concept, the partition manager explicitly stores and maintains the partition mappings. To provide a lightweight memory consumption, tables having the same number of partitions form an equivalence class for which holding a single partition mapping suffices. When the system is scaled using the elastic scaling feature, all partition mappings are updated by computing the optimal load balancing while minimizing the number of reassigned partitions.

Buffer matching mechanism

Scaling an Actian Avalanche installation involves the reassignment of partitions, leading to storage access to perform operations on the partition's data. As a result, the first queries after a scaling process are heavily impacted by the storage access that is significantly slower than accessing buffered data in the main memory. Especially for the case of upscaling the added nodes show substantially slower performance than other nodes. The buffer matching mechanism handles this problem and therefore provides the user with increased system performance after a scaling process. The main idea is to exploit the fact that the system already has information about important data in its buffers before the scaling process. After partitions are reassigned, the mechanism brings already buffered data into the buffers of nodes that become responsible for new partitions.

The buffer matching mechanism is designed as a two-step approach. During the first step, data that needs to be exchanged is selected. Therefore, nodes that lose responsibilities for partitions scan their buffers. For each data block, they determine wether they are still responsible for the block using information provided by the catalog and the partition manager. If they are not responsible for a data block anymore, the block is selected to be exchanged. The second step covers the actual data exchange. Each node sends data to nodes that became responsible for it using a point-to-point communication. To accelerate the process and to reduce the need for synchronization, the data exchange is realized in a multi-threaded manner. Upon data reception, nodes insert received blocks into their buffer while the blocks are removed from buffers of the senders to free buffer space.

The mechanism is integrated in the scaling process. Being an optional optimization, buffer matching is designed to be fault tolerant to not influence the success of the actual scaling process. If an error occurs during the data exchange in a point-to-point communication, this communication is aborted and all data blocks that are not completely transfered are flagged to be loaded from storage instead of using the buffer content.

Evaluation

Our experiments showed a major improvement in the runtime of the scaling process, accelerating the time to the first query result after scaling the Avalanche cluster by a factor 2 to 4, compared to the “inelastic” way of scaling using a system restart. These experiments are highly dependent on the actual cluster configuration and used hardware and do not take resource provisioning time into account, as this may vary for different cloud service providers. Nevertheless these results prove the qualitative improvement of the scaling process. Although the system is not able to handle concurrent transactions during the scaling process, these experiments also show that the system’s downtime for scaling is significantly decreased using the elastic scaling feature. Furthermore the feature offers opportunities for improvements and future projects like allowing concurrent read transaction, as the system remains in an operational state during the scaling process.

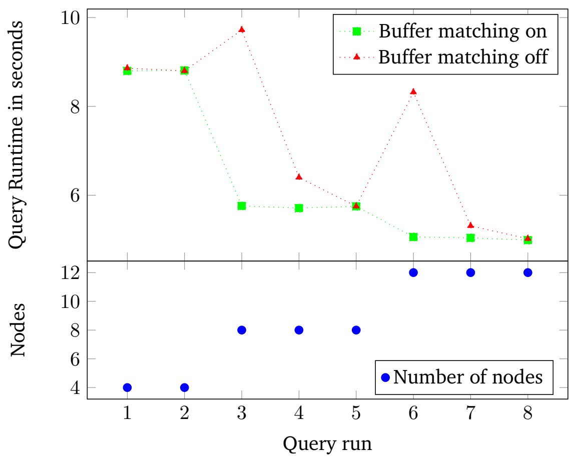

Figure 2: *Scaled query performance after adding nodes*

The buffer matching mechanism also showed to be a major improvement to the query performance after scaling the Avalanche cluster, while also being a trade-off as it adds a minor overhead to the actual scaling runtime but is able to significantly accelerate query performance after scaling. Being an optional optimization, the choice to use the feature lies on the user side.

Experimental results for a basic use case of the buffer matching mechanism is illustrated in Figure 2. In this experiment we consecutively ran a typical analytical query while upscaling the cluster between the runs. When not using buffer matching during the scaling, query performance drops after the scaling process before slightly reaching a better performance in the next runs, which is mainly caused by slow disk access on the added nodes. On the contrary, query performance immediately improves and reaches the minimal runtime when scaling the cluster and using buffer matching, as added nodes already have buffered data for their reassigned partitions and therefore can avoid disk access after the scaling process. As a result, users can immediately benefit from additional resouces they acquire, which is the behaviosr a user expects from scaling the system.

References

Kläbe, Steffen (2019). “Elastic query processing in VectorH”. MA thesis. Ilmenau.