Future Proofing Hybrid Cloud Analytics

Overview

This paper considers how data retained for analytical purposes in a hybrid cloud of geographically distributed data centres can be future proofed against evolving and unforeseen requirements.

Such systems can evolve in a number of ways including, changing data schemas, changes to query patterns or changing data volumes. But any system which requires a change in architecture, change in design or re-location of data to cope with these variations cannot reasonably be considered as future proof as the incumbent system has effectively been replaced by another.

This paper considers the aspects which promote or detract from the ability to future proof a hybrid cloud analytical system.

Dynamic Schemas

One area of change is the structure of the data itself – that is the data schema.

A common approach to handling changes to data schemas is to avoid using a schema altogether. This schema-less approach is common in NoSQL and Hadoop systems where data is acquired in documents or files and a schema is superimposed on top of the data at query time only. Compared to a relational database, this makes the data less open to discovery as there is no schema to explore. For example, consider whether it would it be easier for a user to explore the contents of data presented as a relational schema or as a collection of documents of indeterminate structure. Indeed, the raison-d'etre for a relational schema is to make the semantics of the data transparent and unambiguous. Unstructured documents and files are ideal for containing unstructured data, but much less so for data which has structure to it.

Machine generated data (such IoT device data) is typically structured so that a schema naturally applies to it. For example, machine generated JSON messages contain known paths and keys which naturally map to columns in a message table.

To cope with changing schema requirements, the schema simply has to be dynamic. For example, with a dynamic relational schema, it is possible to add, alter or drop logical structures, such as tables, columns and views, instantaneously and at will. These logical structures reflect the structure of the data and the schema simply evolves with that structure.

Even the most rudimentary relational databases provide dynamic schemas – but only at trivial data volumes[*]. But as volumes grow, most relational databases necessitate the use of performance structures such as indexing, partitioning and de-normalised tables. It is these structures which require design work and take time and resources to deploy; and it is these structures which make relational systems resistant to change – not the relational schema itself.

[*]MySQL appears to be a notable exception here as it apparently rebuilds an entire table even when you simply add a column to it, so that MySQL would not be agile even at relatively small data volumes.

Schema-less systems can still be attractive compared to a dynamic schema if that schema needs to be changed in multiple locations – such as in a hybrid cloud system. If each change needs to be implemented separately at every location, then change becomes difficult and inconsistencies are inevitable while each change is in progress.

In a hybrid cloud system with a dynamic schema it becomes essential to be able to submit each change precisely once from a single location, with a guarantee that the schema remains universally consistent across all locations. Without this ability, the schema becomes resistant to change and it loses its dynamism.

Performance Walls

Performance structures like indexing, partitioning and de-normalised tables are the root of most performance issues – because the correct structures are missing; or because the wrong structures are being used; or because the correct structures are in place for particular queries, but degrade performance elsewhere.

Moreover, changing these structures at large data volumes is not something done very easily. For example, retrospectively adding an index on a table with even just a few terabytes of data will likely take hours and will consume huge amounts of IO bandwidth and memory; while changing the partitioning key of that same data will likely take even longer.

Hence making changes to these performance structures is not done lightly. Worse still, these design decisions will optimise performance in specific areas and degrade performance elsewhere (there really are no free lunches). Hence performance design erects performance walls where the database operates effectively within those walls but offers less than adequate performance outside of them; and it becomes increasing difficult to move those walls as the volumes grow.

Both relational and NoSQL databases are bound by their performance walls and these walls detract from future proofing.

Moreover, changes to performance structures become even more problematic in a hybrid cloud environment, where performance requirements will likely differ at each data centre because of variations in data volume and data population.

Query Agility

Query agility is the ability of the database to respond to arbitrary queries with good response times without prior knowledge or design for those queries. Again, at trivial volumes, relational databases readily provide this agility – but as volumes grow either response times lengthen unacceptably or necessitate the use of performance structures like indexing to maintain reasonable response times.

Even without changing requirements, query agility is important for data exploration. For example, consider a business intelligence tool such as Tableau™. These tools will automatically fire summarization queries as you build your visualizations, to gather information about the data population. These tools will then let you aggregate across large volumes of data and subsequently drill down into the granular detail that lies behind any particular aggregation category. Most database systems fail to service both classes of queries well. For instance, column stores excel at

aggregation queries – but perform poorly when drilling into the granular detail that lies behind an aggregation result.

NoSQL systems struggle too at large volumes because they require design or hardware to yield decent performance. Document stores are fundamentally sub-optimal for analytical queries because their structure is akin to a schema-less row store; while also necessitating indexing for more selective queries. Meanwhile, Hadoop exerts an enormous hardware footprint for even relatively minor data volumes and is not without its own design dependencies – just think about the design choices required around file formats and partitioning.

Table joins can be a performance killer at large data volumes and many relational and NoSQL systems rely on the use of de-normalized tables so that table joins can be avoided altogether. But relying on the avoidance of joins severely limits which queries are possible and constrains data exploration options. Future proofing entails enabling arbitrary queries on demand - so an inability to support arbitrary joins between tables clearly contradicts that requirement.

Summary

A future proof hybrid cloud analytical system may either be schema-less or may provide a dynamic schema which can vary at will and which is managed as a single globally consistent schema across all data centres. But dynamic schemas naturally fit structured data and are more open to data exploration than schema-less systems.

Query agility at scale is also critical, simply because it is rarely possible to see around the corner to make all of the right design choices ahead of time; and the use of performance structures to deliver response times, erects performance walls which are themselves resistant to change. Performance structures become even more problematic in geographically distributed systems where requirements likely differ between data centres because of variations in data volume and data population.

These requirements are very demanding and, in reality, few database solutions are able to fulfil them.