Working Group Report on Cloud Data Services

Chair: Sailesh Krishnamurthy and Fatma Ozcan

Scope: Sub-topics (not exhaustive and based on survey) that should be addressed

- Multitenancy
- Serverless, PaaS, Auto-scaling
- Hybrid Cloud
- Implications of Cloud for Database Engine Architectures
- Auto-Tuning and Usability
- Open source

Introduction

What are the problems that users are experiencing?

Over the last decade, there has been a major shift in data services delivery from on-premise solutions to cloud data services. The major drivers for this move include the elasticity offered by the cloud as well as the ease of start-up time. The cloud environment is fundamentally different from the on-prem environment in that customers expect it to scale elastically. This means scale-out, not scale-up, is the architectural pattern most attuned to the cloud. Rethinking database systems from the ground up to address this is important. In particular, both transaction processing and analytic systems need to be thought afresh from this perspective. Cloud data services are growing as a result of new workloads as well as migration of existing enterprise workloads.

These services are highly flexible and provide users with many options from the nature of provisioned hardware, the types of data services, the ability to decouple storage from compute, the ability to auto-scale the configuration etc. However, this flexibility is often accompanied by high expectations in terms of usability and unlimited scalability that cannot always be met. As we move towards PaaS cloud services we need to double down on auto-everything. It is especially important because already, the simplifications (event the modest ones in current services) are attracting less sophisticated database users who are domain experts, changing our target audience.

These services are built using on-demand compute and storage resources. While this approach enables serverless and pay-for-use models, it also poses challenges for performance predictability, and cloud data architectures.

Some examples of what users want include:

1. Performance and predictable costs without complex knobs or complicated system sizing.
2. Pay-per-use pricing models as opposed to pricing based on pre-provisioned limits
3. Auto-scaling service capacity based on dynamic needs
4. Supporting multi-tenant SaaS applications
5. State management for new idioms like Serverless and Functions-as-a-Service
6. Applications that operate across multiple nodes, multiple datacenters (regionalized), multiple regions (globalized)
7. Applications that operate across on-premise and multiple cloud services seamlessly

Our Accomplishments/Score Card so far

<u>What is the database community already doing to address these problems?</u>

From a commercial perspective, there have been many offerings by vendors of on-prem systems, public cloud services, as well as startups who operate services across multiple clouds. This led to different kinds of data services (NoSQL, Relational, Document, Graph, Analytics, Data Warehouse, etc), some of which are only available as a managed service in the cloud and others which were originally offered as self-managed open/closed source software and now offered as a managed service. The innovation is not limited to offerings by vendors of data services but also driven by engineering teams of large internet services who invent solutions to their ~~very~~ unique problems. In fact some of these solutions are released as open source software which then is eventually offered as a managed service by a startup and/or public cloud vendor.

Although there is a lot of variety in these offerings that together represent progress in each of the problems described in the earlier section, the flexibility in how these offerings can be used has led to significant complexity for practitioners.

Auto-tuning and self-managing databases have been major accomplishments of the database community in the past. While the existing work was more focused about building better query optimizers and recommending materialized views and indexes, there has been lot of newer work in this area that use techniques from machine learning and apply them to database configurations. While this is a step in the right direction it’s hard to argue that the usability challenge has really been addressed.

Influential work on multi-tenancy includes core RDBMS functionality (container databases), customized tools and layers on top of an RDBMS as well as shard management frameworks that are commonly used with open source databases. There has been some amount of application work on multi-tenancy since the early days of Salesforce with two distinct approaches. On one end of the spectrum, all the tenants share one single database and all the application tables. In this case, some of the database functionality has to be pulled into the application. For example, Salesforce has a query optimizer service above the database and heavily uses hints to direct the database optimizer, as it has more information than the database. It is very complicated to support customization for individual tenants with this scheme. Schema evolution and performance isolation are also challenging. However, at least when this approach was new it was economically cheaper. At the other end of the spectrum is separate database instances with each tenant provisioning their own separate data service instance. This is the least cost effective option, but provides the most flexibility for customization, schema evolution and performance isolation. Cloud data services have reduced costs of operation enough to make this approach very feasible. There are also other hybrid approaches. Sharding by the tenant id seem to be one of the most common techniques. Another approach to multi-tenancy is to use the same service instance but isolating tenants by using multiple database schemas. Predictable performance is still a challenge with these approaches.

While serverless is hot today, managed NoSQL services on the cloud have always been serverless. Some Analytics services and DBs have also provided serverless usage models and this trend will increase as data storage in the cloud moves to data lakes. However, serverless relational databases for OLTP workloads have not yet become mainstream and are only used in limited use cases.

The cloud has also significantly changed the architecture of database engines. Whether the database is offered in serverless fashion or not, on the cloud users expect elastic scale up/down/out of database compute and such systems typically separate compute from storage leading to the network as a primary bottleneck in database performance. The disaggregated storage model and often higher storage hierarchies of the cloud create challenges for cloud data architectures, and put a lot of emphasis on caching. Secondly the cloud has changed the fundamental design constraint from dealing with resource scarcity for provisioned environments to resource abundance in multi-tenant environments. Third, elastic provisioning opens the door to novel implementations of core capabilities, e.g., “infinite” write-ahead logs, thereby simplifying the architecture and ultimately, reducing the number of knobs to tune.

Hybrid cloud is the model where enterprise workloads work on both on-prem instances as well as the public cloud. Enterprises still operate a significant fraction of their workloads on-premise due to several reasons. ~~One of the~~ Two reasons for this ~~is~~ are data sovereignty and security concerns for highly sensitive data. Hybrid cloud offers enterprises an option to keep some of their legacy applications on prem, while developing new ones on the cloud, and provides an on-ramp ~~path~~ to migrate their legacy applications to the cloud over time. The biggest challenge for the hybrid cloud is the ability to seamlessly run applications using both on-premise and cloud compute instances.

Public cloud providers are also reacting to data sovereignty statutes by investing in additional data-centers in more jurisdictions. Data sovereignty, business continuity and the need for services that are available worldwide are driving a need for globalized data services in addition to regionalized offerings.

We identified the following **discussion points**:

- Predictable performance: How can we provide predictable performance given the cost-performance trade-offs in auto-scaling?
- Can we converge on a single definition of serverless?
- How does the disaggregated storage, a rich storage hierarchy (including NVRAM) impact cloud architectures?
- Can we use storage disaggregation and metadata integration as a means to stitch together multiple analytic and transactional subsystems into a more unified data management ecosystem, potentially with shared workload management and governance features that cut across the subsystems?
- How do we seamlessly operate across on-premise and cloud architectures with predictable performance? How do we run queries that span across?
- How does the data sovereignty impact data center replication and HA?
- Can open-source and standards help in providing portability between cloud vendors? What are the services, APIs that can be standardized across the various clouds ?
- How can we provide global data services with high throughput and low costs ?
- How to provide schema evolution in a multi-tenant environment?
- How can we use more ML/DL for auto-tuning cloud data services to eliminate DBAs? Optimal data layout (sharding and partitioning decisions), data caching decisions (where and which data to cache), better query optimization, admission control, etc.
- Is there a way to determine when it is beneficial to move to the cloud ? There is a distinction between cost of services vs cost to operate. Is this about cost of ownership, cost of startup, cost of burstiness/elasticity ?
- How can we run analytics workloads across data centers? How about active-active OLTP across data centers? Can we also enable HTAP?
- What type of migration tools are needed to move on-premise workloads to the cloud? Can we maintain same performance metrics?
- How about resource management? How do we leverage shared compute resources?

Call to Action: What should the database community be doing to address these problems?

What should be the goal of the database community? What are non-goals?

Some goals that we identified:

- Usability and predictable cost and performance

- The ability to auto-scale and having proper metrics to study the tradeoffs between cost and performance
- Seamlessly operating on premise and the cloud
- Auto-tuning of the data services, and eliminating the various knobs for performance
- Although usability and predictable cost and performance is a challenge in the cloud, the flip is transparency in both dimensions (cost and performance). Can the cloud catalyze a new era of open benchmarks without the need for auditors etc ?
- Defining a “serverless database” / “serverless data service”. Is it about a cost model (paying on a per-query/per-second basis as opposed to a provisioned basis) ?
- Thinking in terms of what is driven by modern SaaS applications
    - Handling schema evolution
    - Supporting multi-tenancy and sharding
    - HTAP and light-weight reporting on OLTP services

Relationship to other Research Communities

<u>Discuss how the database community should manage relationship with other communities</u>

a. New hardware
    - Cloud data architectures tend to decouple storage from compute in order to maximize elasticity. Does the performance curve change when storage class memory is more viable ? Does the decoupled storage-compute architecture lead to performance challenges ?
    - What happens when storage/memory hierarchy becomes deep ? {block storage - object storage (S3) - cold storage (glacier) }

b. Security and Privacy
    - Multi-tenant vs Single-tenant: Tradeoff between expressiveness and security/confidence
    - Foundational infrastructure to separate out aspects of data processing for execution in a “safe zone”

Sub-Topics

- First solution: Salesforce model 15 years ago; shared tables for tenants, separate type specific tables for indexing, application level optimization logic, heavy use of optimizer hints
- Separate schemas per tenant; easier for development, and performance isolation, but if too many small tenants, too many schemas and the catalog can get too big
- Sharding is common as well, pack tenants to shards, and give large tenants their own shard, providing easy development, and performance isolation

- Hybrid solutions that give large tenants their own schemas (or shards) and pack many small ones into shared schemas

Cloud Data Architectures

- More resources available than on-prem
- Data stored in shared object storage, available to all nodes
- HA and DR must be addressed

Hybrid Cloud

- Seamlessly running the database engine across the public cloud and on-prem systems.
- Need to go from containers/vms running on the public cloud to containers/vms running on prem
- The ability to move computation between the cloud and on-prem servers
    - Data sovereignty needs to be addressed: Data cannot leave certain boundaries

Serverless

- Many different definitions, and we call everyone to converge on a single definition as a result of this study
    - One definition: Serverless computation allows one to code without thinking about hardware/network configuration
    - No need to save state
- Scale up and down easily