

Cloud Data Services: Workloads, Architectures and Multi-Tenancy

Other titles in Foundations and Trends® in Databases

Data Provenance

Boris Glavic

ISBN: 978-1-68083-828-2

FPGA-Accelerated Analytics: From Single Nodes to Clusters

Zsolt István, Kaan Kara and David Sidler

ISBN: 978-1-68083-734-6

Distributed Learning Systems with First-Order Methods

Ji Liu and Ce Zhango

ISBN: 978-1-68083-700-1

Cloud Data Services: Workloads, Architectures and Multi-Tenancy

Vivek Narasayya

Microsoft Research

viveknar@microsoft.com

Surajit Chaudhuri

Microsoft Research

surajitc@microsoft.com

now

the essence of knowledge

Boston — Delft



Documents

Document Sharing Hub

Foundations and Trends® in Databases

Published, sold and distributed by:

now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:

now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

V. Narasayya and S. Chaudhuri. *Cloud Data Services: Workloads, Architectures and Multi-Tenancy*. Foundations and Trends® in Databases, vol. 10, no. 1, pp. 1–107, 2021.

ISBN: 978-1-68083-775-9

© 2021 V. Narasayya and S. Chaudhuri

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, mechanical, photocopying, recording or otherwise, without prior written permission of the publishers.

Photocopying. In the USA: This journal is registered at the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923. Authorization to photocopy items for internal or personal use, or the internal or personal use of specific clients, is granted by now Publishers Inc for users registered with the Copyright Clearance Center (CCC). The ‘services’ for users can be found on the internet at: www.copyright.com

For those organizations that have been granted a photocopy license, a separate system of payment has been arranged. Authorization does not extend to other kinds of copying, such as that for general distribution, for advertising or promotional purposes, for creating new collective works, or for resale. In the rest of the world: Permission to photocopy must be obtained from the copyright owner. Please apply to now Publishers Inc., PO Box 1024, Hanover, MA 02339, USA; Tel. +1 781 871 0245; www.nowpublishers.com; sales@nowpublishers.com

now Publishers Inc. has an exclusive license to publish this material worldwide. Permission to use this content must be obtained from the copyright license holder. Please apply to now Publishers, PO Box 179, 2600 AD Delft, The Netherlands, www.nowpublishers.com; e-mail: sales@nowpublishers.com

Foundations and Trends® in Databases

Volume 10, Issue 1, 2021

Editorial Board

Editor-in-Chief

Joseph M. Hellerstein

University of California at Berkeley
United States

Surajit Chaudhuri

Microsoft Research, Redmond
United States

Editors

Azza Abouzied
NYU-Abu Dhabi

Gustavo Alonso
ETH Zurich

Mike Cafarella
University of Michigan

Alan Fekete
University of Sydney

Ihab Ilyas
University of Waterloo

Andy Pavlo
Carnegie Mellon University

Sunita Sarawagi
IIT Bombay

Editorial Scope

Topics

Foundations and Trends® in Databases publishes survey and tutorial articles in the following topics:

- Data Models and Query Languages
- Query Processing and Optimization
- Storage, Access Methods, and Indexing
- Transaction Management, Concurrency Control and Recovery
- Deductive Databases
- Parallel and Distributed Database Systems
- Database Design and Tuning
- Metadata Management
- Object Management
- Trigger Processing and Active Databases
- Data Mining and OLAP
- Approximate and Interactive Query Processing
- Data Warehousing
- Adaptive Query Processing
- Data Stream Management
- Search and Query Integration
- XML and Semi-Structured Data
- Web Services and Middleware
- Data Integration and Exchange
- Private and Secure Data Management
- Peer-to-Peer, Sensornet and Mobile Data Management
- Scientific and Spatial Data Management
- Data Brokering and Publish/Subscribe
- Data Cleaning and Information Extraction
- Probabilistic Data Management

Information for Librarians

Foundations and Trends® in Databases, 2021, Volume 10, 4 issues. ISSN paper version 1931-7883. ISSN online version 1931-7891. Also available as a combined paper and online subscription.

Contents

1	Introduction	2
1.1	Workloads	2
1.2	Manageability	4
1.3	Multi-Tenancy	5
1.4	Consumption models	6
1.5	Scope and outline of this article	7
2	Cloud Data Services: Workloads and Architectures	9
2.1	Data center architecture and implications	10
2.2	Online Transaction Processing (OLTP)	13
2.3	Data Analytics: Data Warehousing and Big Data Systems .	21
2.4	NoSQL	32
2.5	Observations	34
3	Multi-Tenancy: Background	37
3.1	Virtualization Technologies in Cloud Databases	38
3.2	Observations	43
4	SLAs and Pricing Models	46
4.1	SLOs and SLAs	47
4.2	Factors driving SLAs	48
4.3	Resource-level SLAs	49
4.4	Observations	54

5	Resource Management	56
5.1	Cluster management	57
5.2	Moving Databases Across Nodes	60
5.3	Resource Scheduling	62
5.4	Performance Isolation	64
5.5	Resource estimation	68
5.6	Observations	71
6	Efficiency and Cost	72
6.1	Improving Efficiency	73
6.2	Auto Tuning, Monitoring and Tracing	76
6.3	Observations	80
7	Serverless Databases	81
7.1	Challenges for Serverless Databases	82
7.2	Commercial Serverless Cloud Data Services	84
7.3	Research Proposals	87
7.4	Observations	89
8	Open Problems and Conclusion	90
8.1	Concluding Remarks	90
8.2	Open Problems	91
	Acknowledgements	94
	References	95

Cloud Data Services: Workloads, Architectures and Multi-Tenancy

Vivek Narasayya¹ and Surajit Chaudhuri²

¹*Microsoft Research; viveknar@microsoft.com*

²*Microsoft Research; surajitc@microsoft.com*

ABSTRACT

Enterprises are moving their business critical workloads to public clouds at an accelerating pace. Cloud data services for Online Transaction Processing (OLTP), Data Analytics and NoSQL are essential building blocks for enterprise applications. Multi-tenancy is a crucial tenet for cloud data service providers that allows sharing of data center resources across tenants, thereby reducing cost. In this article we review architectures of today's cloud data services and identify trends and challenges that arise in multi-tenant cloud data services. We survey techniques that have been developed for enabling elasticity, providing SLAs, ensuring performance isolation and reducing cost. We review the emerging paradigm of serverless databases and point out opportunities and challenges. We identify open research problems in the fast-changing landscape of cloud data services.

Vivek Narasayya and Surajit Chaudhuri (2021), "Cloud Data Services: Workloads, Architectures and Multi-Tenancy", Foundations and Trends® in Databases: Vol. 10, No. 1, pp 1–107. DOI: 10.1561/19000000060.

1

Introduction

The worldwide public cloud database services market is large and growing rapidly. The wave of cloud adoption is being driven by digital transformation projects within enterprises that are migrating their applications to the cloud. According to one market study (MarketResearch, 2019), the cloud database services market is estimated to grow from around USD 12 billion in 2020 to around USD 24.8 billion in 2025. Furthermore, according to a Gartner report in 2019 titled "The Future of the DBMS Market is Cloud" (Gartner DBMS Future, 2019), a large percentage (around 68%) of the growth of the overall database market in 2018 came from cloud databases. This report also estimates that by 2022 around 75% of all databases will be deployed or migrated to a cloud platform. Major cloud vendors worldwide that offer public cloud database services include Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform, Alibaba Cloud, and IBM Cloud.

1.1 Workloads

Cloud database services have been developed to meet the diverse needs of enterprise applications. An overview of the classes of database services available in the cloud are shown in Figure 1.1. Relational online

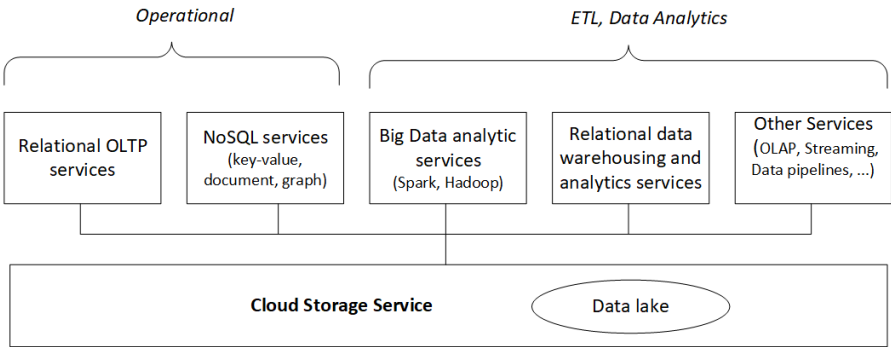


Figure 1.1: Cloud database services

transaction processing (OLTP) services enable enterprises to run their operational workloads spanning multiple industries including ATMs and electronic transfers in banking, reservation systems for airlines, shopping carts and sales in retail, and inventory control in manufacturing. OLTP services are characterized by ability to provide data consistency, high throughput, and high concurrency. Examples of cloud data services geared predominantly towards OLTP workloads are Amazon RDS and Amazon Aurora (Verbitski *et al.*, 2017), Azure SQL Database and Azure SQL Hyperscale (Antonopoulos *et al.*, 2019), and Google Cloud SQL and Google Cloud Spanner (Bacon *et al.*, 2017).

While SQL remains a popular language and is widely used, a class of NoSQL cloud data services have also gained popularity in the last decade. These services cater to applications that need to primarily store and query unstructured data using non-relational data models such as key-value, document, and graph. Examples of such NoSQL cloud data services include Google BigTable (Chang *et al.*, 2008), Amazon Dynamo DB (DeCandia *et al.*, 2007), Azure Cosmos DB (A Technical Overview of Azure Cosmos DB, 2020), MongoDB Atlas (MongoDB Atlas, 2020) and Apache Cassandra.

Data analytic workloads enable enterprises to derive actionable insights from their data (Chaudhuri *et al.*, 2011). Extract-Transform-Load (ETL) services enable transforming data from operational and external sources to prepare for use by analytic services. The industry is seeing

a convergence of data warehousing and Big Data platforms towards a *data lake* architecture, where data is stored on low cost blob storage service in formats optimized for analytic query processing. This data is analyzed with elastic compute nodes using different query processing engines. All major cloud vendors now support the data lake architecture. Services such as AWS Redshift (AWS Redshift, 2020), Azure Synapse (Azure SQL Data Warehouse, 2020), Google BigQuery (Google BigQuery, 2020) and Snowflake (Dageville *et al.*, 2016) support relational data warehousing workloads in the cloud. Enterprises also use Big Data services such as Spark (Zaharia *et al.*, 2010), which uses the MapReduce paradigm popularized by the Big Data compute engine Hadoop (Apache Hadoop, 2020)). Spark is used for diverse workloads such as ETL jobs that prepare data for analysis, and for data analytics using and statistical and machine learning. Finally, there are several other cloud data services such as online analytic processing (OLAP) that supports ad-hoc interactive analysis over multi-dimensional data, and analytics over streaming data that enables near real-time decision making.

1.2 Manageability

In the on-premises setting, the customer, typically an enterprise, is responsible for provisioning and managing the entire stack of both hardware and software needed to run their database. The stack includes server machines, storage, networking, virtualization technology such as virtual machines (VMs) and containers, operating system, database management system (DBMS) and data. Although, the cost of hardware alone in on-premises is perceived to be cheaper than in the cloud, the total cost of ownership taking into account the manageability costs often make the cloud more attractive for enterprises.

Cloud database services offer improved manageability compared to on-premises databases. One option for customers is to provision a VM in the cloud, with a pre-installed image of the DBMS and run their workload in the VM. Durability of data stored in cloud storage services is guaranteed. This takes away the burden of managing hardware, VMs, and storage. Cloud providers are increasingly offering more *managed*

versions of cloud database services. These manageability enhancements include patching and upgrades of the operating system and DBMS, ensuring high availability and disaster recovery of the database, automated backups, and point-in-time recovery and some facets of performance tuning.

Software-as-a-service vendors such as Salesforce and Microsoft Dynamics, that build their ERP and CRM applications on top of cloud database services, further elevate the degree of manageability provided to customers. In such SaaS applications, customers have no access to the underlying databases. While SaaS providers heavily rely on cloud database providers for manageability, they ultimately bear the responsibility of managing databases including any aspects not supported by the cloud database. For example, the SaaS provider is typically responsible for all aspects of performance monitoring and tuning.

Finally, a more recent trend aims to bring the benefits of improved manageability of the cloud platforms to on-premise databases. Platforms such as Amazon RDS for VMWare and Azure Stack allow enterprises to run database instances and storage in an on-premises environment, but manage administrative tasks such as provisioning, software installation, patching, backup and monitoring using the same control plane technology used in the public cloud platform.

1.3 Multi-Tenancy

In on-premises databases the customer owns all hardware resources in their enterprise data center, and hence the databases. In contrast, public clouds are *multi-tenant*. Multi-tenancy is crucial since dedicating hardware for each tenant is simply not cost effective. In a multi-tenant cloud data service multiple databases, potentially from different customers (a.k.a. *tenants*), share computing resources of the cloud provider including CPU, memory and disk resources of a machine and the data center network.

Cloud providers virtualize the available physical resources of a machine such as CPU, memory, disk I/O, network I/O, and local storage into logical units (e.g. VMs, containers). Each tenant's database executes within such a logical unit of virtualization. This provides two

major benefits to the tenant and the cloud provider. First, virtualization typically offers a certain degree of security and performance isolation between tenants. For instance, virtualization technology may enforce resource governance mechanisms to reduce impact of a noisy neighbor on the performance of a tenant. Second, virtualization of resources is the key to enabling multi-tenancy since multiple databases can now be consolidated onto a single machine (node) while still ensuring security and performance isolation. Consolidation is *crucial* for enabling cloud providers to lower the cost of providing the service, in particular by reducing the capital expenditure (CapEx), since increased consolidation means they need to purchase fewer servers to serve the same number of databases. Thus, multi-tenancy, is a core tenet of cloud databases, and indeed cloud computing.

For a cloud database provider, a fundamental design choice is determining what virtualization technology to use so as to suitably balance security, performance and cost considerations. In practice, they employ a variety of technology ranging from VMs, containers, operating system processes, logical containers within a DBMS process, to even sharing tables within a single database. The choice, in turn, has a big impact on the quality of service, how resources are managed and the cost of running the service.

1.4 Consumption models

The most widely used consumption model in the cloud are *provisioned* databases. At the time the database is created, the tenant specifies a fixed set of resources that is promised to the database. The tenant pays for the fixed set of resources whether or not they use them. Such a consumption model is attractive for enterprises workloads since they offer the assurance that resources are always available.

More recently, cloud providers have started to offer *serverless* databases, which offer a different consumption model. Tenants no longer need to provision resources ahead of time. Rather, the cloud provider acquires and releases resources in response to demands of the database workload, and tenants only pay for resources that their workloads actually use. To enable good performance for serverless databases at low cost, cloud

providers need to develop resource management techniques that are elastic and efficient.

1.5 Scope and outline of this article

Today's data center architectures, multi-tenancy and novel consumption models bring new requirements that do not exist, or are not as important, in on-premise database systems. These requirements have led to new technical challenges and significant changes in the software architecture of databases.

1. *Quality of Service and Pricing:* Customers of cloud database services may have varying needs of quality of service for performance, availability and cost. There are different kinds of service-level objectives (SLOs) that cloud service providers strive to achieve. For some services, providers guarantee an SLA for availability or performance. Similarly, there are different models of pricing of database services. These SLAs and pricing depends on several factors including multi-tenancy.
2. *Resource management:* Multi-tenancy requires that the resources in a datacenter be managed so as to ensure that each tenant achieves the performance and availability SLOs despite sharing resources with other tenants. This gives rise to technical challenges in managing clusters of machines, performance isolation, resource estimation, resource scheduling, and placement and migration of databases within a cluster of machines.
3. *Cost:* The success of cloud data services crucially depends on cost of goods sold (COGS) for the cloud provider and the total cost of ownership (TCO) for the tenant. As a consequence, several techniques have been developed such as resource harvesting and resource overcommitting, which reduces capital expenditure (CapEx), service intelligence, which reduced operating expenditure (OpEx), and auto-tuning functionality, which reduces resource consumption and TCO for tenants.

4. *Security*: Multi-tenancy in cloud database gives rise to challenges in security such as preventing malicious tenants or system and database administrators from gaining access to data.

This article focuses on state-of-the-art public cloud data services for Relational OLTP, Relational and Big Data Analytics, and NoSQL workloads. In Chapter 2, we review these workloads, and discuss the architectures of modern cloud data services, drawing contrast among services across multiple cloud providers and academic research where appropriate. In Chapter 3 we provide the background of different models of multi-tenancy that have been adopted by cloud databases. Chapter 4 focuses on the quality of service guarantees offered in commercial cloud data services and novel proposals from the research literature. We also discuss pricing options and the opportunities that they enable. In Chapter 5 we discuss the challenges and solutions for the key issues in resource management: cluster management, performance isolation, resource scheduling, and resource estimation. In Chapter 6 we review techniques for reducing cost for customers and providers of cloud data services through techniques such as overcommit, resource harvesting and auto-tuning. In Chapter 7, we review the emerging paradigm of *serverless databases*, which aim to raise the level of abstraction for provisioning and elasticity in the cloud along with pay-per-use billing.

We have made an attempt to discuss techniques developed both in industry as well as academic research. In each chapter of this article, we conclude with our observations and a set of open technical challenges on that topic. In Chapter 8 we conclude with a discussion of open issues that go beyond the specific topics discussed in individual chapters. We note that although security in cloud data services is an important topic in its own right, it is beyond the scope of this article.

2

Cloud Data Services: Workloads and Architectures

We review cloud database architectures that have emerged in the industry to support the following widely used database workload categories, and the implications of these architectures for provisioning, performance, scale, and cost.

- Online Transaction Processing (OLTP)
- Data Analytics: Relational data warehousing and Big Data Analytics
- NoSQL (non-relational): Document and key-value stores

The diverse functionality, performance, and scale requirements of each of these workloads types, combined with server and networking characteristics in modern data centers, lead to substantially different architectures for services. Indeed, even within a workload category, different cloud providers sometimes adopt different architectures.

Examples of these services whose architectures we discuss in this article: OLTP services such as Amazon Aurora, Azure SQL DB Hyper-scale, Google BigTable and Spanner, NoSQL services: Mongo DB Atlas, Azure Cosmos DB, Amazon DynamoDB, Data analytic services such as: AWS Redshift, Azure SQL Synapse, Snowflake, and Google BigQuery.

2.1 Data center architecture and implications

2.1.1 Regions, Availability Zones, Data Centers

Cloud providers cluster their data centers into physical locations around the world, called Regions. Within a single Region, there are often multiple sub-clusters of data centers called Availability Zones (AZs). Each AZ has its own redundant power and networking. AZs are physically separated from each other by meaningful distances, but within a latency-defined perimeter (e.g. few 10's of miles). All AZs within a Region are interconnected with high-bandwidth, low-latency networking. Networking speeds are usually fast enough (usually a few milliseconds) to enable synchronous replication between AZs. Partitioning an application across AZs can therefore allow an enterprise to increase protection from issues such as power failure and even certain kinds of natural disasters. Latencies *across* Regions are typically much higher, often in the order of 100s of milliseconds, and vary significantly based on geographical proximity.

2.1.2 Data center servers and networking

It is common for each server to have two or more sockets, 10s of physical cores per socket, 100s of GB to a few TBs of RAM and 10s of TBs of local SSD. Each new generation of servers is different, and typically increases the number of cores, RAM and SSD available per server. One important implication of having a large amount of resource per server is multi-tenancy: it is vital that the service provider be able to share those resources across multiple tenants since most tenant databases require only a small fraction of each of the resources on the server. Therefore, it is necessary to virtualize the resources of the physical machine and parcel those resources across tenant databases. We discuss the types of virtualization technology used in cloud data services in Chapter 3.

The traditional data center architecture used in on-premises data centers of large enterprise is shown in Figure 2.1. The network is a hierarchy with racks of servers at the leaf, and a layer of core routers at the top. The servers in a rack are connected to a top-of-rack switch (ToR). ToRs are connected to two aggregation switches for redundancy, which in turn are connected to a higher layer of aggregation

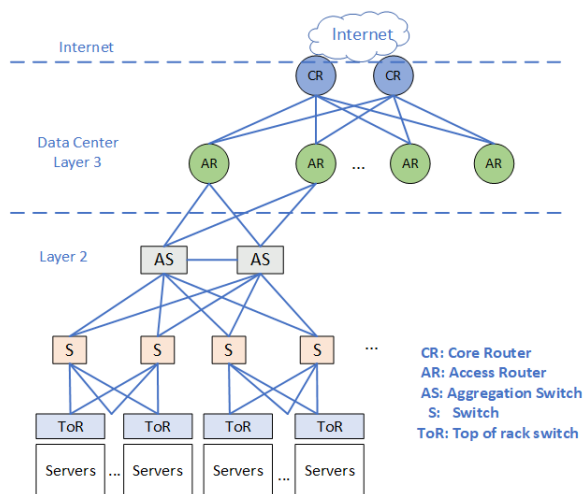


Figure 2.1: Conventional data center architecture

switches and access routers. The core routers are responsible for carrying traffic between access routers and to manage traffic in and out of the data center. A key limitation of this architecture is that server-to-server bandwidth is limited due to the fact that the higher layers of the network are heavily oversubscribed. This has serious consequences for cloud data services. Most data in a data center resides in storage services. Cloud OLTP, Data Analytic and NoSQL services need to access data in storage services over the data center network. Furthermore, in data analytic services, compute nodes need to exchange data such as intermediate query results with other compute nodes. Therefore, server-to-server bandwidth limitations directly impacts performance and scalability. To better support cloud data services and other public cloud services including machine learning and high-performance computing workloads that require higher server-to-server bandwidth, data center networking architectures have undergone significant changes over the past decade.

Modern data centers used by public clouds adopt a different network architecture, using relatively low cost commercial switches and arranging them into a Clos or fat-tree topology (Dally and Towles, 2004) which provides extensive path diversity between servers. An example of such

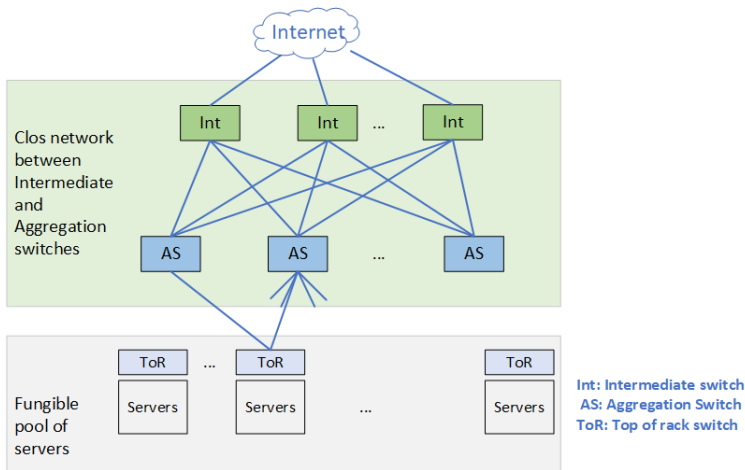


Figure 2.2: Data center architecture using Clos Network

a network architecture is shown in Figure 2.2, where the links between the Intermediate switches and the Aggregation switches form a bipartite graph. Therefore, between any two servers in different racks, there are many possible paths. Thus, failure of a single device has a small impact on bisection bandwidth. Furthermore, by using routing techniques such as equal-cost multi-path routing (ECMP) and randomization, e.g. Valiant Load Balancing (Valiant and Brebner, 1981), traffic can be spread across these links thereby avoiding bottlenecks. A more in-depth justification of data center networking architecture and design decisions can be found in these references: (Al-Fares *et al.*, 2008), (Greenberg *et al.*, 2009) and (Alizadeh and Edsall, 2013).

2.1.3 Implications

Hardware trends and network data center architectures have heavily influenced the architectures of modern cloud data services. In a traditional monolithic database engine architecture both compute and storage are coupled on the same node. High availability and durability in the presence of node failures are achieved by running multiple secondary databases that serve as hot standbys. This results in good performance but high cost.

However, storage services in the cloud are often much cheaper when compared to local SSDs on compute nodes. Furthermore, a storage service replicates a data file across its servers thereby offering durability and high-availability of data files independent of the computer tier. This availability and durability at scale is a central tenet for cloud data service providers. Combined with large bisection bandwidths available in the data center network, it has driven the evolution of *disaggregated* architectures for OLTP, data analytic cloud and NoSQL services. In this setting, compute nodes can be spun up or down and re-sized based on workload demands. Such independent scaling of the tiers depending on the workload demand also helps lowers cost. All local storage on the compute tiers are treated as ephemeral and their primary role is in caching data. Failure of compute nodes may affect performance but not durability of data.

Despite advances in networking architectures and technology, cloud data services can still face challenges in achieving high performance due to network bandwidth limits and round-trip latencies. Data analytic services shuffle large amounts of data across compute servers. Similarly, the latest generation of cloud OLTP services need to move data over the network from storage to caching layers to compute nodes, and between compute nodes. Thus, for both kinds of services, effective data caching at the compute tier, both in-memory and on local SSDs becomes crucial for ensuring good performance.

2.2 Online Transaction Processing (OLTP)

The Cloud OLTP services we discuss below support SQL APIs over a relational data model. To better understand their unique strengths and how they compare with one another, we consider a few important dimensions: how they achieve **high-availability**, **durability**, **scalability** with data volume, their **performance** characteristics, and **cost**. We first review OLTP services that like the traditional, on-premises architectures for OLTP, couple compute and storage, but are hosted in the cloud. These services can achieve excellent performance and high-availability, but are limited in their ability to scale to large data volumes and incur high cost. Next we review the new generation of cloud-native OLTP ser-

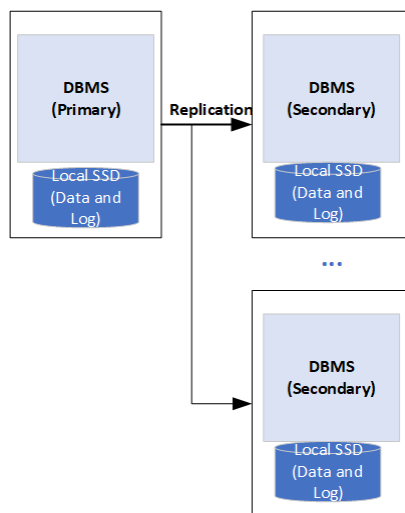
vices are built on the principles of separating the compute layer, which is responsible for processing SQL queries and providing transactional semantics, from the storage layer that is responsible for providing fast access to consistent data pages and providing durability. These services aim to achieve high-availability at scale while keeping costs low.

2.2.1 Architecture with coupled compute and storage

Azure SQL Database Business Critical and Amazon RDS are examples of cloud OLTP services where the database runs in a DBMS process called the *primary* server. To achieve high availability, these services use log replication with one or more *secondary* servers which behave as hot standbys (see Figure 2.3a). The secondaries can support read-only queries. Upon failure of a primary, one of the secondaries is elected to become the primary, and takes over. A new secondary is spawned with a copy of the data to replace the failed server. To achieve high-availability of 99.99 or higher, typically three or more secondary database servers are required. We observe that durability is achieved through storage of the data and log files on each of the primary and secondary servers.

There are a few key points to note about these services. First, the architecture follows the familiar paradigm in on-premises databases for achieving high-availability using replicated state machines. Second, in this architecture high I/O performance, both in terms of latency and bandwidth, is achieved through the use of SSDs for data and log files. Third, the scalability of such services is ultimately limited by compute and storage capacity of a *single* node. The amount of SSD storage that is available can be a limiting factor on the size of databases that can be supported (e.g. only a few 10's of TBs). Finally, since multiple secondaries are required for achieving high availability, the data needs to be replicated to SSD storage of each secondary, thereby sharply increasing the storage cost.

Despite the data scale limitations discussed above, many enterprise OLTP applications require high performance and operate at data volumes well below the above limits. Furthermore, cloud vendors allow customers the choice of how much compute resources (CPU, memory and IOPS) to provision, which can be used to partially control costs.



(a) DBMS with high-availability using replication

Figure 2.3: OLTP data services with coupled compute and storage

Therefore such cloud OLTP services remain viable for a segment of customers.

Finally, there are alternatives to using log replication for achieving high-availability in cloud databases with coupled compute and storage. Amazon RDS Multi-AZ (Amazon RDS Multi-AZ, 2020) uses block replication functionality at the Linux kernel level to copy modified data blocks at the primary node to the data volume of a secondary node in a different availability zone (AZ). Google Cloud SQL provides Persistent Disk service (Google Persistent Disk, 2020) to achieve high-availability for database applications. It offers an SSD option for supporting high-performance database workloads and performs synchronous block-level replication at the storage layer. Data blocks modified by the primary are replicated to storage at the secondary. Azure's Shared Disk provides capabilities similar to Google Persistent Disk. For database systems it is important that each of these alternative solutions are zero-RPO (Recovery Point Objective) i.e. in these systems no data is lost when the primary goes down because replication is synchronous. However, synchronous replication is practically limited by the geographic distances

due to latency of a round-trip, and is therefore typically suitable for servers within the same region.

2.2.2 Architectures with disaggregated compute and storage

The scalability and cost limitations of co-located OLTP cloud database services have led to a new class of OLTP architectures, where compute and storage tiers are separated. This enables the storage and compute tiers to scale independently of one another. Achieving high-availability of the overall service requires each tier to be highly available. Since these services store the durable version of data in relatively inexpensive storage the cost of durability is reduced when compared to storing multiple replicas of the database on local SSDs. However, since the entire database is no longer available locally on each compute node, aggressive caching is needed in the compute and storage tiers to offset the increased latency of data access. In this section, we review and compare three such scale-out cloud OLTP services: AWS Aurora, Azure SQL Hyperscale and Google Cloud Spanner.

AWS Aurora

The architecture of AWS Aurora (Verbitski *et al.*, 2017) is shown in Figure 2.4. The compute nodes run the standard logic of a database engine such as transactions, query processing, access methods and caching. To achieve high-availability of the compute layer, Aurora uses a primary compute nodes that handles both read and write transactions, while multiple secondary nodes exist that can process read-only transactions. Multiple secondary compute nodes can be spawned depending on the read load. However, in contrast to the architecture described in Section 2.2.1 where the transaction log from the primary is sent to the secondaries, in Aurora logging is handled by a custom-built storage service. The transaction log from the primary is also sent directly to the secondaries *in addition* to the storage service in order to keep the buffer pool of the secondaries warm and reduce replica lag.

The storage service is a distributed system that stores multiple copies of the data across availability zones to achieve high-availability. When a compute node requests a page, the storage service returns the

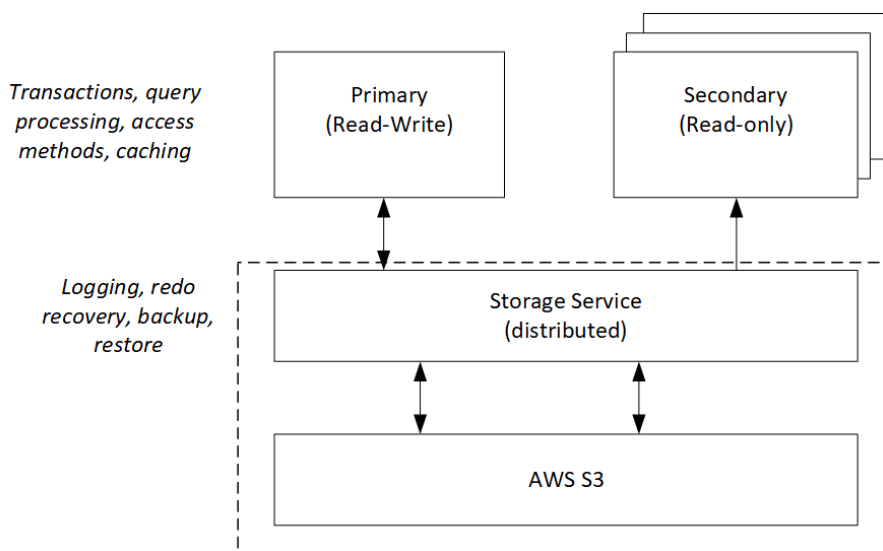


Figure 2.4: AWS Aurora architecture

current version of the page by applying redo log records as needed to bring the page up to date. This application of log records can be done asynchronously and does not fall into the critical path of query processing on the compute node. The compute nodes as well as storage service caches data on local SSDs to improve performance. Traditional implementations that leverage distributed storage would use distributed consensus algorithms for commits, reads, replication, and membership changes and amplify cost of underlying storage. Aurora avoids the high performance penalty typically associated with distributed consensus (e.g. 2PC) in most circumstances by establishing invariants and leveraging local transient state (Verbitski *et al.*, 2018).

Azure SQL Hyperscale

The architecture of Azure SQL Hyperscale (Antonopoulos *et al.*, 2019) is shown in Figure 2.5. While it has similarities with AWS Aurora 2.2.2, we focus here on some of its unique aspects. Azure Hyperscale is decomposed into three components: a compute tier, log service tier,

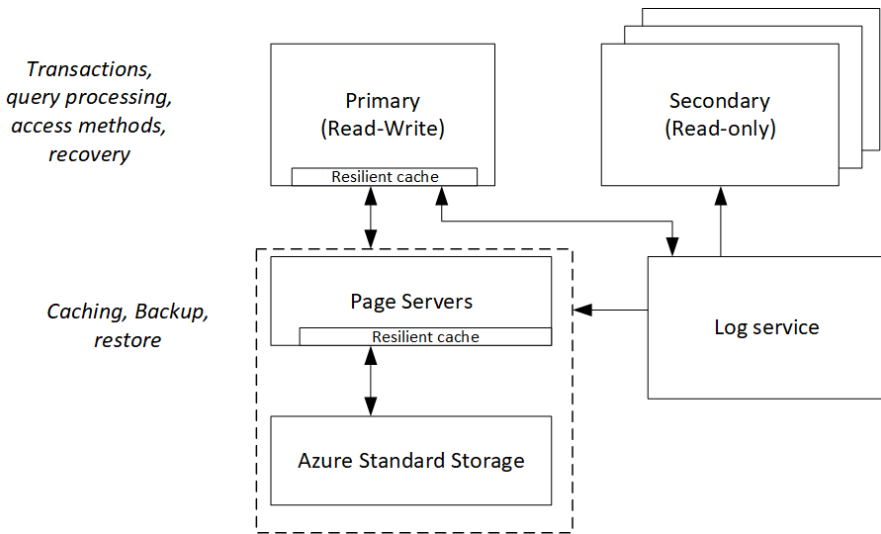


Figure 2.5: Azure SQL Hyperscale architecture

and a storage tier. Unlike AWS Aurora where logging is handled by the storage tier, Azure Hyperscale uses a separate, custom-designed *log service*. The log service is designed using multiple replicas and a quorum based protocol, thereby reducing the time to make the log durable which is on critical path of transaction commits. Further, the log service leverages access patterns of log records by keeping recent log records (which are usually in high demand) in main memory and scales by distributing them across multiple nodes. Older log records, which are needed less frequently (e.g. to abort a transaction) are staged to secondary storage.

The storage tier consists of *page servers* and Azure Blob storage. Each page server holds a copy of a partition of the data, and plays two roles. First it serves pages to compute nodes by applying log records obtained from the log service. To optimize performance, the page server caches the entire partition on the local SSDs and the warm data is cached in the memory of the page servers. One advantage of caching the entire partition on local SSD is that even if there is an intermittent loss of availability of Azure Blob Storage, database availability is unaffected. Second, page servers also checkpoint the data and create backups of the data on Azure Blob Storage.

Google Cloud Spanner

Cloud Spanner is a scale-out, globally distributed and synchronously replicated database (Bacon *et al.*, 2017). Like AWS Aurora and Azure SQL Hyperscale, Cloud Spanner supports the relational data model and SQL queries. However, unlike AWS Aurora and Azure SQL Hyperscale, which are *shared disk* systems and write transactions are processed only by the primary server, Cloud Spanner is a *shared nothing* system. To achieve scale with data volume, data from each table is sharded (i.e. horizontally partitioned), and within a data center shards are distributed across multiple servers. To achieve high-availability, each shard is then replicated to multiple, geographically separated data centers. Write transactions in Cloud Spanner can span multiple shards.

An overview of the Cloud Spanner architecture is shown in Figure 2.6. A given replica stores data in an append-only distributed file system called Colossus (Corbett *et al.*, 2013). The storage is based on log-structured merge trees. Cloud Spanner's transactions use a replicated write-ahead redo log. Each replica applies the log to its copy to bring it up to date. The Paxos consensus algorithm is used to get replicas to agree on the contents of each log entry. Concurrency control is achieved using a combination of pessimistic locking and timestamps. One of the key enablers for managing timestamps in a geo-distributed setting is the TrueTime API, which at any point *now* returns a conservative time range within which *now* must fall. Truetime leverages GPS and atomic clocks to derive conservative ranges on the current timestamp used in concurrency control.

Since data is sharded and geo-distributed, Cloud Spanner uses a distributed query processing engine on top of its storage tier. This layer performs standard optimizations in distributed query execution such as partition pruning, co-locating shards of different tables involved in a key foreign-key join etc. Cloud Spanner also automatically handles query restarts if one or more nodes fails. Handling transient failures has several benefits including a simpler programming model for applications, ability to handle node upgrades, reducing tail latency, and allowing forward progress for long-running queries.

In summary, Cloud Spanner has a fundamentally different architecture in comparison to AWS Aurora or Azure SQL Hyperscale, and is able

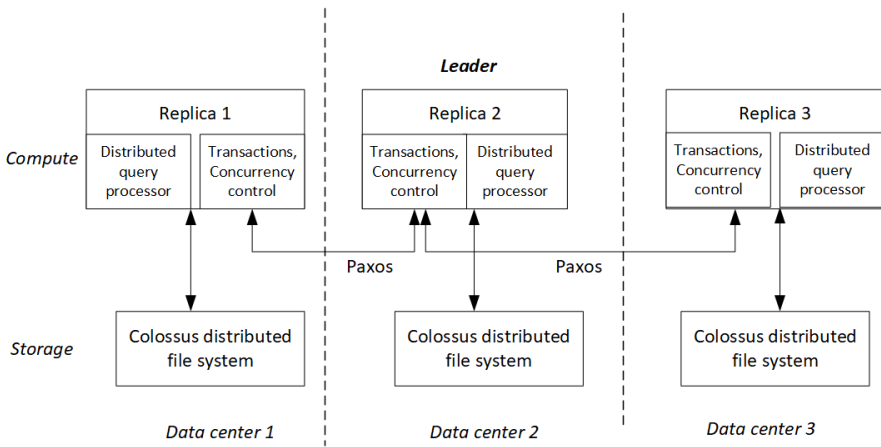


Figure 2.6: Spanner software stack (Corbett *et al.*, 2013)

to support distributed transactions and distributed query processing over geo-distributed data. However, it does incur the cost associated with expensive distributed consensus protocols and data movement. Therefore, for enterprise OLTP applications where the compute resources of a single primary with multiple readable secondaries is adequate to meet the performance demands, AWS Aurora or Azure SQL Hyperscale may be sufficient since they avoid the overheads of distributed transactions and query processing.

Low-cost cloud OLTP databases

In contrast to the architecture of cloud OLTP services described in this section thus far that aim to support mission-critical databases with high performance, scale and availability requirements, there is another class of cloud OLTP services that cater to cost-sensitive database applications. In this architecture too, compute and storage layers are separated. However, they only run a single DBMS process that is attached to data and log files on relatively low-cost remote storage (see Figure 2.7). For instance, in Azure SQL Database General Purpose, customers can use remote storage on Azure Blob Storage. Upon planned or unplanned failure of the database, the DBMS must be restarted, potentially on a

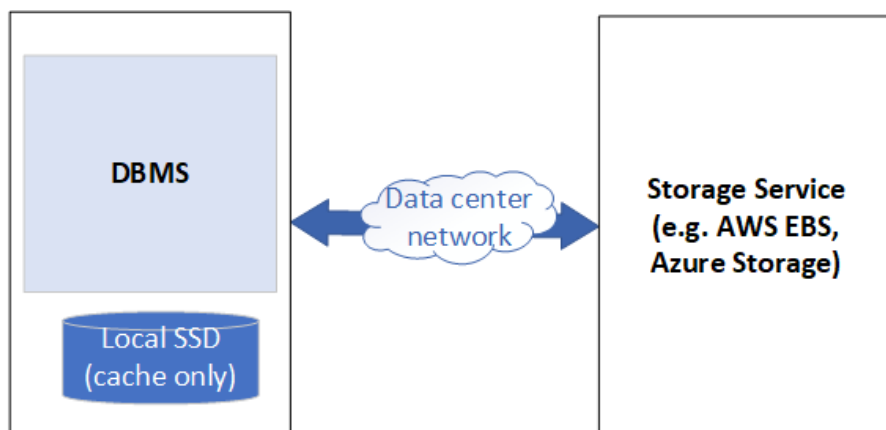


Figure 2.7: OLTP Databases for cost-sensitive applications

different machine. Therefore, in comparison to architectures involving hot standbys, availability is lower. Data durability is assured via replication in the storage service layer. This class of databases also use the local SSD, but limited in size, and only as an optional cache for warm data to accelerate performance. The performance and scalability of such database are ultimately limited by network bandwidth and latency, and the compute power of a single node.

Finally, we note that cloud vendors also provide faster storage options such as AWS Elastic Block Storage (EBS), Google Persistent Disk and Azure Disk Storage which are SSD backed systems. Thus, can trade-off higher costs for improved performance. It is worth noting that databases have consistently driven improvements in storage services in the cloud, and has parallels to similar improvements in prior decades when DBMS vendors develop custom storage solutions (e.g. use of raw partitions vs. file system for Oracle databases).

2.3 Data Analytics: Data Warehousing and Big Data Systems

Businesses rely on Business Intelligence (BI) software for making better and faster decisions (Chaudhuri *et al.*, 2011). A key enabling technology for such decision support software are data analytic services. Data

analytic services fall into two broad categories: *Data warehousing* and *Big Data analytics*. Data warehousing engines typically require data to undergo a loading step before it can be queried, thereby allowing for greater performance optimizations using indexes, materialized views, and data partitioning. Traditional, on-premises data warehousing engines use shared-nothing parallel DBMS technology, where compute and storage are co-located. In contrast, Big Data analytic services allow analyzing data residing in a shared storage subsystem, e.g. HDFS on-premise or blob storage in the cloud in a standard format such as Parquet, ORC, Avro, or CSV, without requiring an explicit loading step. They are typically cheaper and can support analysis on much larger volumes of data than data warehouses.

As highlighted in the Seattle Database Report (Abadi *et al.*, 2020) the needs of data analytics applications have expanded over the years. They often consume data from a variety of data sources and analyze them in-situ without necessarily going through an expensive loading and indexing step. The volume of data has also increased significantly. The industry is seeing a transition from primarily data warehouses to a *data lake*-oriented architecture for analytics. Data arriving into the cloud is stored in the data lake using low-cost, scalable blob storage. A variety of compute engines can operate on the same data, to curate the data (e.g. using Spark) or execute complex SQL queries using a scale-out relational data warehousing engine, and the results are stored back into the data lake. Similar to modern cloud scalable OLTP services, data analytic services in the cloud are also embracing disaggregation to enable independent scaling of compute and storage. This has had a profound impact on the architecture of these engines.

While AWS Redshift initially adopted a traditional shared-nothing architecture where compute and storage were coupled, it has subsequently added options for exploiting disaggregated storage by selectively moving cold data to remote storage. Other cloud data warehousing services such as Snowflake, Google Big Query, Azure Synapse SQL (formerly Azure SQL Data Warehouse) opted for separation of computer and storage tiers from the start. The Big Data system Spark (Zaharia *et al.*, 2010), which is based on the MapReduce (Dean and Ghemawat, 2008) paradigm were also designed with separate compute and stor-

age tiers (HDFS). They targeted applications such as log analytics and extract-transform-load (ETL) jobs that require efficient handling of user-defined functions and user-defined aggregates. More recently, support for SQL queries on top of Spark has started to get increased adoption, particularly Spark SQL.

One interesting trend in recent years is that the line between data warehousing and Big Data analytics services is starting to blur. For example, data warehouses allow querying *external tables*, i.e. a file stored in a remote storage service can be made visible to the data warehouse as a table and can be queried along with pre-loaded tables. Likewise, some data warehouses services now allow the service to pause the compute nodes when there is no load, and subsequently resume them to help customers reduce cost. On the other hand, some Big Data analytic engines are beginning to support performance accelerators, e.g. indexes in Azure Synapse Spark, and cubes for accelerating OLAP queries on top of data in HDFS in atScale (AtScale, 2020). The term *lakehouse* is often used to describe systems that combines the best elements of data lakes and data warehouses. Some of the key elements of a lakehouse are in-situ analysis of structured and unstructured data stored in potentially different open formats in a disaggregated storage service, while providing data warehouse capabilities such as transaction support (e.g., Delta Lake (Armbrust *et al.*, 2020)), reporting and integration with BI tools.

In this section we review and compare several major cloud data analytic services: AWS Redshift, Snowflake, Azure Synapse, Google BigQuery and Spark along a few key dimensions:

- Shared nothing vs. shared data architecture
- Programming API: SQL vs. MapReduce
- Pre-loaded data vs. In-situ querying
- Interactive vs. Batch querying capabilities
- Sophistication of query optimizer

We note that many of these services are available both in a provisioned model – where compute resources are reserved up front and

paid for whether or not a query is executing, as well as via a serverless model – where compute resources are spun up only when the query is executed and shut down after the query completes execution. We defer these discussions to Chapter 7, where we discuss serverless databases.

2.3.1 Snowflake

The Snowflake Elastic Data Warehouse (Dageville *et al.*, 2016) is a cloud data warehousing service that uses a shared storage architecture, i.e. multiple compute nodes access data stored in shared remote storage such as AWS S3 or Azure Blob Storage. It supports a standard SQL API. The architecture overview of Snowflake is shown in Figure 2.8. The data is stored in a cloud blob storage service such as Amazon S3 or Azure Blob Storage. Similar to traditional data warehouses, Snowflake requires data to be ingested (i.e. loaded), before queries can be executed against it. Tables are horizontally partitioned into large blocks and stored as files on remote storage. During data load time, for each partition, the system stores the data in a compressed columnar format and computes zone maps, which are the min and max values for each column in that partition. Additional statistics necessary for query optimization are also collected during data load and stored and managed by a metadata service.

The compute nodes are standard virtual machines VMs in AWS EC2 or Azure Compute. Within each VM runs a database engine designed from scratch. VMs are organized into a collection called a *virtual warehouse* that handles distributed query execution. Multiple virtual warehouses belonging to a single tenant can execute at the same time, thereby allowing elastic scale as the query load changes. Since the data being processed is no longer local, the VMs cache data on local SSD to exploit locality and accelerate performance. Snowflake also supports traditional features in data warehousing such as materialized views that can greatly accelerate performance for repetitive query workloads containing expensive operations such as join, group by and aggregation.

Although shared storage architectures are used in on-premises database systems such as Oracle RAC, Snowflake was one of the first cloud-native data warehousing services to demonstrate viability using

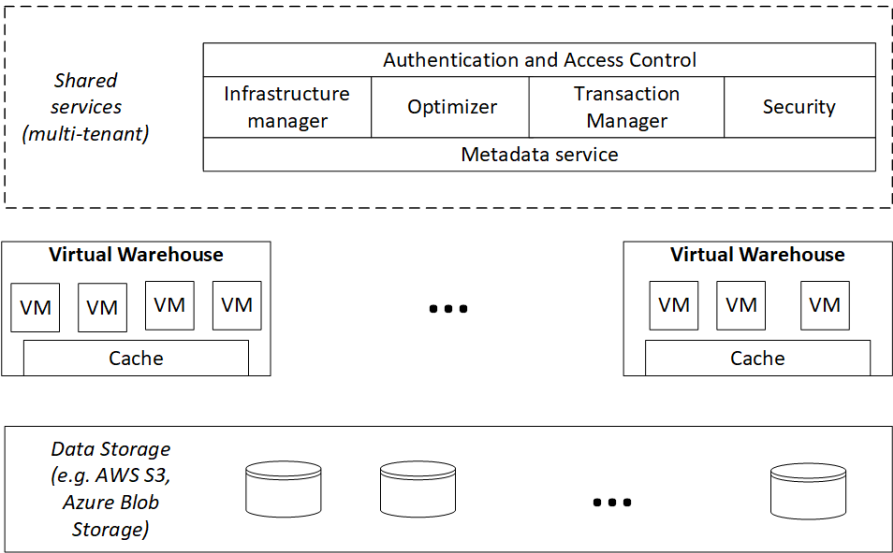


Figure 2.8: Snowflake architecture

commodity cloud compute and low-cost shared cloud storage. It has focused on scalability and ease-of-use by enabling good default options for challenging manageability problems such as data partitioning, and simplifying query processing by avoiding indexes.

The shared services layer of Snowflake is a multi-tenant service and manages virtual warehouses, transactions, query optimization, security etc. Snowflake’s query optimizer follows the Cascade-style approach of top-down cost-based optimization. Because Snowflake does not support indexes, the search space of the query optimizer is reduced compared to one in a traditional relational data warehouse.

2.3.2 AWS Redshift

AWS Redshift follows the classic Massively Parallel Processing (MPP), shared-nothing architecture for data warehousing (see Figure 2.9) and supports a SQL API. A Redshift cluster is a collection of compute nodes and a *leader node*, which coordinates query execution on the compute nodes. The leader node also handles external communication with client application. A cluster contains one or more databases. Data must be

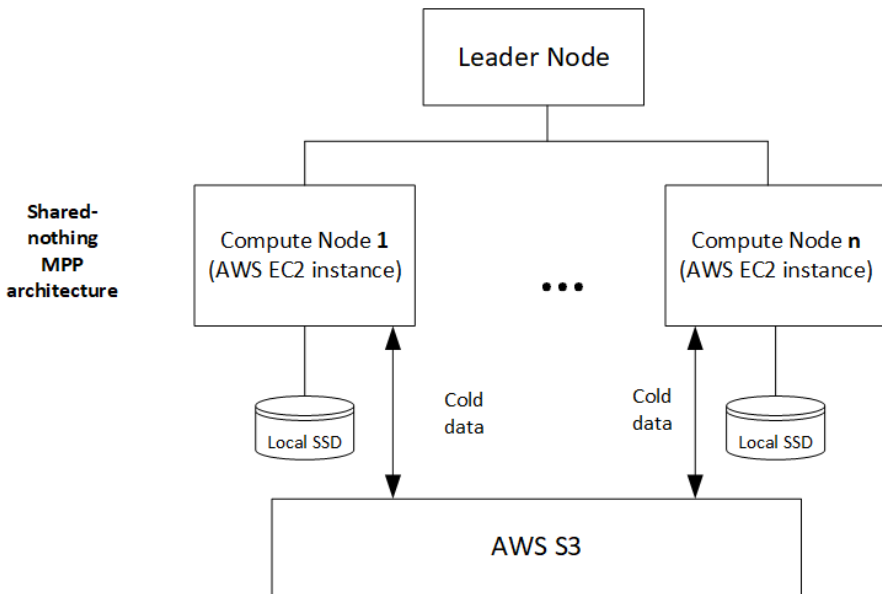


Figure 2.9: AWS Redshift architecture

loaded into the service, and is stored in a compressed columnar storage format. A table is distributed across the compute nodes by specifying the column(s) to use in the distribution key or by choosing that a table to be replicated to all compute nodes. This data is stored on local storage of the compute nodes. While such an architecture is well-established in on-premises data warehousing systems (e.g. Teradata), it has a few limitations, particularly in the context of the cloud. Efficient scale up and down of compute resources based on query load is difficult, since if the number of compute nodes is increased (or decreased), it requires a redistribution of the data, which is expensive and can compromise availability and performance. Second, local storage is more expensive than remote storage in the cloud.

In order to take advantage of the advantages of cheap and durable cloud storage service (AWS S3) and reduce cost, AWS Redshift has introduced new optimizations. It uses new EC2 instances called RA3 for the compute nodes. RA3 nodes have the capability to automatically move cold data from local SSD to AWS S3, while retaining hot data in the

local SSD of the compute node. RA3 uses the AWS Nitro System (AWS Nitro System, 2020), a family of cards that offloads and accelerates I/O functions from the CPU to hardware. This allows moving cold data to and from AWS S3 without imposing a high CPU cost on the compute node.

2.3.3 Google BigQuery

Google BigQuery (Google BigQuery, 2020) is based on the Dremel system (Melnik *et al.*, 2010), which was designed for interactive ad-hoc analytic querying on nested relational data. It supports a SQL API¹ and is a multi-tenant service. A more recent article (Melnik *et al.*, 2020) summarizes new functionality and optimizations that are included in BigQuery. The architecture of BigQuery is shown in Figure 2.10.

BigQuery supports both pre-loaded data as well as in-situ analysis on data stored in file formats such as CSV, JSON or Avro or via federation from other database engines such MySQL. When data is loaded into BigQuery, it is transformed into a custom compressed format, automatically sharded, and written to remote storage (the Colossus distributed file system). The columnar format used, called Capacitor, includes optimizations that enables efficient query evaluation techniques such as skipping partitions that do not contain qualifying rows for the query, vectorization, and skip indexes that allows skipping segments of data within a partition when a predicate is very selective. While columnar formats are widely now widely used in all data analytic engines, Dremel spearheaded the columnar formats for semi-structured data. When a query is executed, the Dremel engine maps the SQL query into an execution tree. The leaves of the tree read the data shards from Colossus. The intermediate nodes perform aggregation and data shuffling (e.g. for joins) moving data across the network. Since the service is multi-tenant, Dremel dynamically apportions resources to queries as needed while maintaining fairness across multiple users.

One important focus of BigQuery is supporting interactive query performance over petabyte scale data volumes. The initial focus was on

¹Both Google Cloud Spanner and Google BigQuery support the same SQL dialect.