Deep Science Publishing
https://doi.org/10.70593/978-93-49910-26-3_5

Chapter 5: Architecting Scalable Cloud Infrastructure to Support High-Volume OSS/BSS Workloads in Real-Time Environments

5.1. Introduction

Cloud computing provides a new impressive model of IT architecture which offers new methods for development, maintenance and management of scalable applications. With the growing interest and acceptance of the Internet economy, as well as the continuous improvements of data storage and computing capacities, more and more IT resources have been set up out of the local Intranet infrastructures. Applications are being developed and re-architected to be deployed and externally hosted in a privately managed or publicly outsourced distribution of data flows, databases and processing nodes around the Internet. Such applications must ensure reliability and availability at affordable costs, efficiency of resource usage and must have small or zero maintenance overhead. This poses new challenges in the design and implementation of such large-scale distributed applications relying on clusters of distributed machines whose performances heavily depend upon the design of the Bus and of the interconnection links (Andrikopoulos et al., 2014; Botta et al., 2016; Erl et al., 2017).

In order to support real-time, high-volume workloads, Cloud infrastructures have to deal with multimodal streams of big data and to ensure automated business continuity of long transactional or analytical processes. This must be done at an affordable cost for infrastructures whose characteristics change with time. The distributed geolocation of Cloud clients also poses technical challenges for transactions needing real-time interactions with sensitive content. These requirements cannot be met by commercial Clouds. These limitations may be overcome at a limited extra cost by designing and implementing a dedicated Cloud infrastructure shared by a small number of clients. Applications have to rely on a large number of different software tools for managing heterogeneous sets of heterogeneous physical or virtualized machines and for enabling a "Business Operations Cloud." This document presents the design and implementation of a cloud architecture able to satisfy our Clients expectations. We propose a "one size fits all" architecture which presents a small set of innovative design points explicit in the rest of the text (Khajeh-Hosseini et al., 2010; Ghosh & Naik, 2016).

5.1.1. Overview of the Document Structure

The purpose of this document is to investigate how communication providers can architect their IT operations in the cloud for real-time OSS/BSS workloads; understand the scaling characteristics of the components across the spectrum of an OSS/BSS system; enumerate their building blocks and design their IT operations around them. Each individual scaling characteristic can be property modelled and accounted for, and the entire OSS/BSS stack not only work in the cloud, but also are as efficient or more efficient than any other on-premise IT structure. The various service models available in the cloud service spectrum can be leveraged, accounting for differences and nuances and using them as levers to reduce infrastructure costs while increasing operational efficiency. Chapter 1 introduces the contents of this document. Chapter 2 provides context around the methodology for simulating and modelling OSS/BSS workloads. Chapter 3 provides insights into telecommunication workloads, both proprietary and open source, and discusses the testing methodology and the associated logging. Chapters 4 to 6 discuss how these workloads look like architecturally. Chapters 7 to 9 provide recommendations into architecting real-time OSS/BSS operations from a cloud architecture point of view. The merits of doing transport or packet processing in both the service mesh layer and the orchestration controller layer, the requirement for virtualization, the need for latency and scheduling in making resources in backend and database layers are discussed. An exhaustive scaling characteristics model for OSS/BSS workloads is then presented based on our exhaustive measurements. The remaining chapters of the document would explore available choices for those who wish to architect private OSS/BSS clouds. Chapter 10 describes the internals of the various components of a private cloud, such as region, availability domain, and physical and logical networks, and describe how to deploy these components in a redundant, fault tolerant fashion. Finally, Chapter 13 provides final thoughts on architecting clouds for real time OSS/BSS workloads.

5.2. Understanding OSS/BSS Workloads

1. Definition of OSS/BSS

As many of the telecommunications companies create a multitude of new offerings in the face of increased competition, Operations Support Systems (OSS) and Business Support Systems (BSS) components have become immensely critical in supporting these high-volume telecommunications transactions. OSS/BSS perform functions such as order processing, inventory management, billing and provisioning support and as a result are subject to incredibly high-volume transaction processing requirements. Moreover, with increased competition and customer desire for instant gratification on services/products requested, customers are becoming more and more intolerant of delays in the order processing cycle.

In recent times, more and more telecommunications activities including call detail record processing, credit card processing and business reporting decision support systems processing run under real-time conditions as a result of the increased use of advanced computer systems architectures in the telecommunications business.

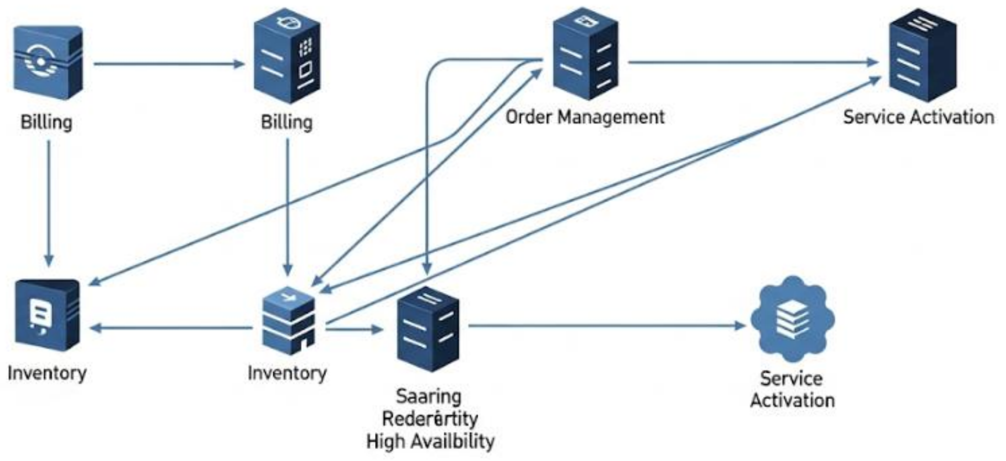

Fig 5 . 1 : OSS/BSS High-Volume Transaction Processing

2. Characteristics of High-Volume Workloads

The extreme characteristics of high-volume OSS/BSS workloads could be summarized as follows: time to customer receipt of service; demand for service availability; operational personnel productivity; service portfolio; nature of error processing; handling of special desires; demand for reliability; transaction response time requirements; zero backlog for transactions in process; communication support; customer/product profitability analysis; transaction processing distribution; requirements for parallel processing; growth rates; software development productivity; investment; and return acceleration.

3. Real-Time Processing Requirements

The increased concerns on the part of the telecommunications customer with regard to their ability to receive immediate online response time to any inquiries have terminated the data research and data mining functions carried out by the telecommunications company. In summary, the desire of telecommunication customers for quick resolution of all telecommunications transactions ranging anywhere from wrong billing amounts to service upgrades has dictated that all activities surrounding these types of transactions must be carried out extremely quickly and efficiently.

5.2.1. Definition of OSS/BSS

Operations Support Systems (OSS) and Business Support Systems (BSS) are essential components of the telecommunication world, supporting virtually all operations, from providing and managing services to charge and bill customers. Modern telecommunication systems such as mobile, fixed, cable or satellite networks have two types of major components, the core and the edges. The communication service providers charge customers for the use of the edge networks, and then what services are delivered using them. The services delivered, powered by the networks, define the content of the business, while the temporal, financial and performance attributes persuade customers to acquire such services.

The core network, responsible for the transmission of all traffic, can also be a major item of discussion between the CSP and the content providers. However, no major works are generally based on charge policy or revenue share setups for the use of core infrastructures although it could come to attention considering emerging situations, such as multi-access edge computing frameworks. The space of advertisement and marketing presents promising areas of focusing works in this area. Revenues are the motives for all players, neither end users nor CSPs are interested in dealing with technology aspects. Such issues with technology need to be simplified and redesigned to be put in the background. OSS/BSS systems exist as pillars of the CSP business enabling the management of technical issues in a way that is transparent for customers. After dedicating the technology chapter to OSS/BSS external and internal architectures, this chapter will focus on the details of processing workloads. The words architecture, system, model, and framework will be alternatively used just for the sake of writing fluency, as they have proper definitions.

5.2.2. Characteristics of High-Volume Workloads

High-Volume OSS/BSS Workloads require defining additional parameters and constraints in order to specify the management of high-volume Workloads in general. The workload parameters that are considered will include job-parameter ranges, workload activity distributions, generalization of concurrent-user activity, debugging output, security, availability and reliability, large databases, and volume of user services. Constraint definitions will include maximum job throughput times; maximum job response times; transaction-user service-time parameters; on-line real-time transaction-processor security and convenience service methods; and maximum on-line real-time transaction-user service times. Some of the items need fairly specialized approaches, and if, in spite of this, they do not appear to be fully treated, they must be added in the judgment of the System Architect. Constraints may certainly depend on the type of service being provided; again, the actual data processing is part of the Communications Architecture.

Service-level objectives defined by Management often depend on external constraints, while the available data to ascertain SLO-meeting or -breaching is usually based purely on historic operability data analysis of certain of the system Behavior Parameters, available over a sufficiently long series of months. This data is totally insufficient to successfully and agreeably justify a total Service-Level Approach. The type and relative amount of Work Load is, of course, the predominate factor in externally visible Security, Availability, and Responsiveness SLO-Parameters. The management decision on each of these parameters usually implicitly defines the entire possible Constraint-Parameter Space, and the given decision for each SLO is made primarily for reasons of user convenience or importance of external prestige.

5.2.3. Real-Time Processing Requirements

Most of OSS and BSS functions have been created to provide near real-time or real-time information. OSS services roll up all relevant events from the communication network, aggregate them into kind of a summary, and store this summary in a temporary storage. For example, alarm services monitor the quality of services, report service degradation, traps alarm analysts, prepares decisions on service rebooting, issue commands to the network for serviced impact mitigation, and data for the problem analysis. In the event of anomaly deviation, these services are expected to dispatch alarms to specified alarm analyst workbenches in less than a minute. All these services must process thousands of events in a minute, which is a scalability problem. Also, the delay from the network servicing problem occurrence, analysis of the task to be done, and execution of remedial task must not exceed 10 minutes. Typically, problem analysis is handled on dedicated analytical systems, but the availability of real-time analysis in cooperation with other OSS/BSS systems have been requested, especially in large networks. Composite the events, and provide pre-packaged solution to the NOC to avoid unrest among subscribers.

BSS systems collect customer data from different operational agents and store them in a dynamic, multilevel, multidimensional data repository. The maintenance of this data repository has to be near real-time. The customer data specification must be devoted enough to ensure that the identification in the case of a particular customer is sufficient, and data preservation is acceptable in case of large number of transaction delays. Unauthorized customer identification must be able to be performed by rules based on customers' preferences, social behavior, and event-related usual behavior.

5.3. Cloud Infrastructure Fundamentals

This section provides a concise introduction to basic cloud principles and concepts, which lay the groundwork for the rest of the discussion in the report. It includes an overview of the cloud computing service models, important infrastructure components that allow realizing virtualization, and selected comparative data for leading cloud service providers, which offer their services to the public.

1. Overview of Cloud Computing Models

The delivery of on-demand service burst size from the public internet is what differentiates cloud computing from other distributed computing models. Cloud computing offers customers on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. Cloud service customers may select an operating system, programming language, web application platform, database, storage, and all associated applications from a cloud service provider's offering catalog and configure software packages such as antivirus and security-monitoring for operation in the cloud. Cloud service customers can interface with various basic types of service model and service architecture. The stronger separation from underlying resources, the higher level of abstraction achieved, the greater interruption of services during migration, and the increase in complexity for secure interoperation with internal enterprise services distinguish the types of public cloud services from one another.

Infrastructure as a Service (IaaS) is the cloud model that provides virtualization containers and payloads as the basic managed offering. Typically, IaaS customers use virtualization to create multiple bounded instances of an operating system environment on the same physical server machine. IaaS is the lowest cloud model layer. For this reason, IaaS has less built-in abstraction and provides customers with the greatest control over allocated underlying resources compared to other models. However, IaaS entails the highest service migration costs in terms of time, complexity, and risk from unavailability or data exposure when compared to other operating models. The introduction of Platform as a Service (PaaS) as a higher level cloud architecture allows customers to focus on building applications rather than managing and controlling cloud infrastructure resources used to execute applications.

5.3.1. Overview of Cloud Computing Models

In this chapter, we discuss fundamentals of cloud infrastructures, including some technologies and services. We outline key cloud providers and components of their offerings, mainly in software. We elaborate on cloud computing models and discuss potential bottlenecks and architectural issues in scaling cloud infrastructures onto Big Data.

Cloud computing is an innovative technology that offers new business models for developing and deploying internet scale applications. The cloud computing model is inevitably associated with hyperscale providers of cloud services, ever evolving infrastructure as a service, and large scale deployments of virtualization technologies. Due to economic drivers, including concentration in a few cloud cities, in time-to-market for new services and for migrating into public cloud, in capital expenses for IT resource purchase and in operational expenses for managing owned infrastructures, it is anticipated that large telco service providers will develop and offer cloud-enabled platforms and infrastructures supporting the next generation of telecommunications services.

To support existing business models, and due to high entry barriers into high-volume OSS/BSS cloud space, service quality requirements for cloud-enabled support of telecommunications are notably more strict than those applicable to general, non-telecom related cloud services. Cloud-enabled telecommunications services fall into three main categories: strong flexibility in adapting to customer specific deployments, rich connectivity and communications capabilities, low latency and high availability requirements for sensitive telecommunications workloads. To address these requirements, telco service providers need to closely cooperate with hyperscale cloud providers, acting as direct customers of cloud infrastructures for supporting strategic infrastructure projects.

5.3.2. Key Components of Cloud Infrastructure

The cloud computing architecture is essentially composed of two basic components: the front-end and the back-end. The front-end is the side of the client and the back-end is the side of the provider. The cloud structure enables the user to manage resources over the Internet via a simple graphical User Interface (GUI) that allows data upload/download, monitor resources, configure applications, and many other tasks related to cloud computing. Also called Client, the front-end consists of the computer, laptop, tablet, or mobile that the user connects through the Internet to access cloud services. The front-end also consists of the applications used to access the Internet and the cloud service provider network and the protocols used to communicate with the back-end. The back-end is the side of the service provider and consists of physical servers or clusters of servers and Internet connection infrastructure that deliver the desired cloud services. According to the type of services provided, the back-end contains other components such as resources virtualization program, network management, data storage, data communication, Database Management Systems (DBMS) and tools, and application development and management tools. The cloud can provide one or a combination of the following services: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), Business Process as a Service (BPaaS), Function as a Service (FaaS), and Container as a Service (CaaS). The IaaS service provides server hardware resources to users. The PaaS service provides infrastructure and tools to the user so that the user can create applications to run on the provider's infrastructure. The SaaS service provides applications that run on the provider's infrastructure, require only user access via a front-end device, and are used for business and office functions like word processing, email, customer relations management, accounting, etc.

BPaaS service is a variation of SaaS which provides a business function service process that is affordable, efficient, reliable, and scalable. FaaS services provide application functions that can be invoked, run, and terminated on a cloud provider infrastructure. CaaS services provide containers that are hosted, executed, and managed on the cloud provider infrastructure. The cloud service management enables users to request and perform tasks and modify a service via a web portal or a web browser without need of low-level programming. On the provider side, the cloud service management executes users' requests and schedule actions for allocated resources.

5.3.3. Cloud Service Providers Comparison

Major cloud service providers have various characteristics. AWS RCUs and WCUs are required to be provisioned by customers. This is a major negative tradeoff and opportunity cost for customers, because there is no data usage monitoring, scaling and billing from DynamoDB. With Google Cloud Firestore, scaling and billing are seamless and the only requirement is to set up billing alerts. However, Firestore charges on the number of document reads, writes and deletes, as well as storage space. Market share leaders in NoSQL are AWS DynamoDB, Azure CosmosDB and Google Cloud Firestore.

AWS DynamoDB and Azure CosmosDB provide managed services that enable gaming and other applications to make millions of requests per second, in low latency, predictable performance. Google Cloud Firestore is optimized for mobile applications. For our use case, we do not need these three services to execute millions of requests per second. Performance should be excellent, and costs should be pay as you go. This constraint makes Amazon Aurora Serverless and Google Cloud Firestore the optimum solutions. AWS Aurora Serverless enables automatically locking, scaling and unlocking according to usage, with Postgres and MySQL compatibility.

Open source databases are not applicable for executing legacy software, due to lack of scaling, availability and other optimization. Ceph is a distributed file system, and there are not open source databases that scale and offer high performance and AI optimization. Additionally, Ceph is based on network attached storage, with a single file system, with single RAID optimization. Google Cloud Firestore, and AWS DynamoDB and Azure CosmosDB, are not only NoSQL, Document Model, Cloud Services. They also integrate AI and provide billions of intelligent services by API.

5.4. Design Principles for Scalability

Designing for performance and reliability is essential in building enterprise-class solutions that are scalable. Each business solution will have varying design principle implementations that fit the problem domain. This section will cover a set of recommended design principles that can be observed in carrying out the design activities independently or collaboratively to arrive at the solution architecture that solves business problems. Scalability is an important architecture characteristic, and it can be achieved through the following mechanisms at various levels in the architecture, such as application, middleware, data, network, and hosting infrastructure.

Scalability can be planned by focusing on key architectural principles, such as decomposing business capabilities into scale-units and then identifying the organic scale-units to be hosted separately; strategically exposing business capabilities as services that can be used with different protocols and technologies with varying Quality of Service characteristics; testing expected loads early in the system lifecycle; observing actual usage and tuning the implementation to meet the preferred Quality of Service; planning for redundancy or using multiple hosting farms for business capabilities that are bound to be bottlenecks; exposure of common-business problems as services; using clusters of similar technology/specialization for hosting redundant capability implementation or bottlenecks; balancing data access to avoid data craft bottlenecks; partitioning data access and data master-slave replication; balancing network and single-point failure planning; and making the environment supportable to manage spikes by tracking surge patterns. Scalability can be defined quantitatively as the additional load on the platform to achieve a certain level of Quality of Service for a group of workloads, over and above the initial load that defines the existing Quality of Service of the same workloads.

5.4.1. Horizontal vs Vertical Scaling

Horizontal scaling, also termed "scale out", is the use of multiple small computers (nodes) to complete a task. In terms of infrastructure, it would imply a cluster of multiple servers that can be used in conjunction or independently to fulfill the workload. The most effective use of horizontal scaling is when architecture can be used to process workloads in bulk. There are, nonetheless, some limitations to horizontal scaling: the necessary underlying architecture to distribute the workload has to be put in place, and any common resource has to be accessed in an appropriate manner, without bottlenecking any nodes. In addition, some tasks are too large or too complex and cannot be easily divided to suit the smaller nodes. Not every software can leverage the advantages of horizontal scaling, especially if it is not designed to do so. In regard to workloads, activities such as rating, charging, and mediation of network data are ideal horizontal-scalability candidates, as they can perform processes on individual data records, or alternatively manage a queue of records.

Vertical scaling, or "scale up", uses one computer to increase resources used by task completion. This is done by adding more processors, memory, or disk space. Ever larger server sizes are available for use, and with the advent of multi-core processors, a single server now has the capacity to take on many more workloads. In principle, large databases can benefit from vertical scaling to optimize response time, as all database requests can be satisfied by a single resource. However, as large corporate customers demand larger quantities of data to be managed by single databases to reduce overall database expense, scaling a database into a monolithic server solution creates potential limits to the longevity of that particular database. There are practical limitations on how far a single server can be scaled, the main one being the increase in expense associated with configuring and maintaining large databases on such a server.

5.4.2. Load Balancing Strategies

The canonical definition of load balancing describes it as the process of distributing requests over a set of resources or servers in such a way that no individual server is compromised sufficiently to degrade performance. When designing a load balancer, there are two design aspects that we need to define: What type of traffic are we balancing, and what core algorithm do we use to make the balancing decision. Two common types of applications that require load balancing are static applications, along the lines of a file server that serves static files, and a dynamic application, along the lines of an application server that serves requests invoking complex logic.

Static content requests are typically I/O-bound with very small and predictable load times. In this case, load balancing acts primarily to ensure that requests are distributed evenly across servers. The core load balancing algorithm can leverage a simple round-robin approach where the load balancer diverts the traffic to every server in a cyclical order. More sophisticated approaches maintain an internal count of the number of requests a server has processed and transmit the traffic to the server that has processed the least. These algorithms are designed to precisely balance throughput across machines. However, traditional load balancing does not take delay into account when determining the allocation of problems across clusters. Backend server loads are not always held equally by all servers in a cluster, nor do they evolve over time in a cyclical fashion. Instead, each server has bursty loads where the problem size changes over time. Further, latency is usually an important quality metric. Load balancing is important for servers processing variable-sized requests, since while loads may be kept equal, latencies may not.

5.4.3. Auto-Scaling Mechanisms

Scaling adjustments typically result from observed changes in workload demand. An administrator tunes the conditions that will cause the scaling operation to occur. This operation is either triggered by specific events or is performed at scheduled times. An auto-scaling policy uses the data gathered from the resource monitoring function to initiate such timely adjustments. On dynamic load-balancing systems, a transfer of the existing virtual machine load to newly provisioned machines and/or executing services running lower priority can occur before scaling in.

Managing the timing of scaling actions is an intricate part of auto-scaling. Scaling delays and the timing of changes to the system directly impact the total completion time of the set of transactions. Scaling actions are susceptible to overheads associated with bringing more resources onto the system or removing resources from the system. These scaling transitions may take time and cause a small wastage of resources. Potentially, they may result in an increased cost for systems that charge for resources on a per-hour basis. Furthermore, it is essential to ensure that scaling in does not remove resources vital for executing existing transactions, as this leads to decreasing the overall throughput or possibly to failures. In practice, simple thresholds do not yield satisfactory performance. Bandwidth-based control has proven a useful enhancement to this approach.

Cloud services scale by the addition and removal of virtual machines, rather than by the addition and removal of actual physical machines. Auto-scaling these virtual machines is especially useful in performance testing of cloud applications because they may be added based on the ramp-up schedule of the application, but the details of the machine count are not usually included in a workload description. The ramp-up is the increase in load with a series of increment valleys, where the actual load is placed in low-priority mode, until the load reaches a transaction queue length average that corresponds to the resource equilibrium wait time. Once steady-state operation is achieved, the load can be maintained at this maximum, again with the actual performance in the low-priority mode.

5.5. Architectural Patterns for OSS/BSS

In this chapter, we summarize some popular architectural patterns for deploying complex systems that are used for the day-to-day functions of OSS/BSS. Deploying real-time cloud infrastructures for OSS/BSS workloads is not trivial. We start by outlining the general architectural patterns for the deployment and use of OSS/BSS in a cloud environment, specifically outlining patterns that are useful for high-volume, high-traffic OSS/BSS workloads. The architectural patterns described here can be used for the deployment of engineering, business, operational, service delivery, security, or virtual OSS/BSS system functions. These architectural patterns are also useful for the deployment of CSPs for IT services that monetize IT workloads in niche markets. The common architectural patterns discussed below are based on microservices, serverless, and event-driven architectures.

Microservices Architecture Microservices is a modular architectural pattern used for deploying cloud-native applications or cloud-hosted services. It provides a set of best practices for enterprise applications in the architectural and service design of network-based client-and-server business solutions that provide diverse cloud-agnostic software. Microservices architecture is based on many loosely coupled components or services, each supporting low-level business operations, that together offer a more complex enterprise application or versatile service. Each service performs a specialized business capability and uses only a small subset of the enterprise or bigger application data. Services are implemented through lightweight programming models and exposed for use as APIs or over multiple messaging protocols. Each service can be designed and built independently of the others, permitting each service development team to work in parallel, using the best language for each service.

Microservices architecture is best suited for enterprise applications with the following characteristics: support micro-deployment to hide the complexity of scaling the overall enterprise application; capable of scaling to support huge volume pressures expected from the enterprise workload; support independent release, rollback, and energy of application logic; deploy independent monitoring of enterprise capabilities and application health; utilize distributed data management techniques for high data throughput rates; designed to accommodate high availabilities; and able to accommodate development teams skilled in specific programming languages for specific microservices.

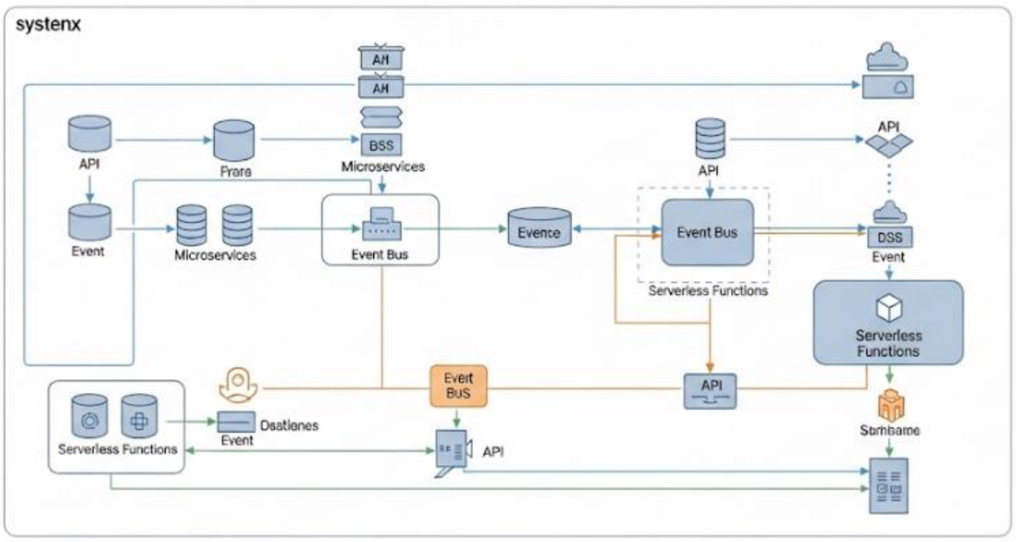

Fig 5 . 2 : OSS/BSS Functions

5.5.1. Microservices Architecture

Microservices architectures are defined as the applications that consist of multiple Microservices, which are small, independent, and self-contained applications that implement a single capability of the larger application. Microservices talk to each other to create a cohesive and functionally complete application, typically leveraging application programming interfaces and lightweight technologies to ensure simplicity. The main advantages of a Microservices architecture are that it supports rapid development and scaling by managing small and specialized teams and operations, and it fosters agility, checkpoints, and parallelization. Microservices architectures are ideal for applications that are scalable, frequently updated, and mission-critical, such as OTT, ecommerce, and other applications.

Modern Agile and DevOps methodologies are aligned and in perfect sync with Microservices architecture design principles. Traditional overhead, bottlenecks, and handoffs inherent to monolithic architecture design and development, such as long QA cycles, staging deployments, lack of segregation, and slow overall release times, are resolved in Microservices designs. Separate and independent Microservices, owned by focused teams, are capable of updating code independently and rapidly deploying changes using DevOps techniques to a cloud-based platform. These small teams can also write, deploy, maintain, and operate the majority of the codebase in the Microservice both during the initial phases of product evolution, and in steady state. Independent Microservices are then stitched together and tested at a higher level of abstraction during an integration and functional QA stage of development to ensure that the collective services perform correctly as a larger product. Problems resolving around complex dependency management issues of team coordination, parallel coding, and releasing of larger products are resolved using Microservices architecture design principles.

5.5.2. Serverless Architectures

Faced with the need to deploy microservice architectures and to architect and build the systems to be scalable, cloud providers have launched their products and tools to support on-demand provisioning of serverless components along with several build, logging, tracing, and deployment services. Serverless is now considered one of the buzzwords in the cloud-native space, and almost all events to do with cloud-native technology or digital transformation mention at least a couple of sessions on serverless architecture. But the question we face is – how is serverless relevant to OSS/BSS? This is not a new question and various service providers have dallied into serverless computing. They have picked up specific workloads along with marketing angles and innovations, published reports, and taken the technology for a spin.

So what are serverless architectures? When a subset of on-demand resources such as elastic compute, storage, and networking in the public cloud ecosystem is combined with event or API gateway technologies to trigger specific events–based processing flow, we are said to have entered serverless. Why is it then considered a stack or API enabling technology? While serverless is an event-level or short time period computation paradigm, microservices act at the service layer and provide long-lasting business functions. Hence while functioning at different levels, they are complementary for businesses considering a cloud-native revolution or software evolution. With different operational characteristics, serverless provides more agility and dynamism but at the same time imposes development and operational constraints. Our advice to OSS/BSS architects is to consider the right set of functions and functions within services or associated service calls that lend themselves to serverless to reduce time to market and set CDI across operations and support teams.

5.5.3. Event-Driven Architectures

Event-driven architectures (EDAs) represent a natural extension of microservices architectures and overlap with serverless and microservices on various dimensions, including building blocks composition, resource provisioning, orchestration challenges, technology stacks offered by public cloud providers, and major services offered by specialized cloud service providers. However, because microservices and serverless have redundancy or similar functionality, serverless and microservices converge with event-driven more often than any of the other components converge with each other. For example, in the example for microservices, the address resolution component's shared inter-computer resources are implemented using a specially-designed microservice middle tier, while an edge cache can also make use of a serverless component added atop the address resolution microservice middle tier.

Both microservices and serverless components share common functions to create an event-driven architecture that matches the functionality found in an enterprise service bus (ESB) which is specially suited for other specific business domains, but often is not architected to use the same distributed message bus that an EDA comprises. Major real-time business domains covered in an EDA, where the incorporated service requests and responses as well as events are available at the resources on which they affect for analysis by the any included ad hoc and batch analytics services, as well as company dashboards, share similarities with the differences of real-time and batch analytics infrastructures. Clients receive real-time push alerts whenever event thresholds, such as event volume or query event count, exceed user set thresholds in the included DDoS detection infrastructure.

5.6. Data Management Strategies

Data management is to OSS/BSS workload what the Concorde is to aircraft: the best solutions in the industry point to no other choice than a selected set of commercial systems. There may be another choice feasible for smaller service providers with standardized products and relatively low volumes: replicate database engines that are in fact original. But for the big ones with unique products, amazing offers, and monstrous volumes, query processing, availability, and data sustainment are no other than commercial database solutions. For the lesser enabled service providers willing to accept the trade-offs imposed by re-inventing the wheels, open-source solutions could do.

Database selection boils down to a few factors: performance, internal architecture, scalability, and replication functionalities. Volume, transaction, and resource requirement in terms of tolerable latency and application throughput will dictate performance. Internal architecture is a vital selection element since it governs scaling properties and capabilities. The functions exposed for data replication are key for all applications needing to interface with the operational database. On the one hand, contributing database solutions must replicate data to still active databases. Meanwhile, other data products would consume and synthesize it, or for some use cases, in real time. These operations, performed with proprietary interfaces or with data streaming tools, have constraints, as bandwidth, query after loading delay, and intercepting redoing logs.

Application and business stakeholders are primarily responsible for the big data solutions in terms of volumes, transactions, and costs, even though they will be using easy to use, no SQL type data management applications. These solutions will point to a selected few solutions; some might have come from the data management niche under the manufacturer's umbrella, where their function is to provide niche advantages for enormous and unique data volumes and considerable but very few use case requirements.

5.6.1. Database Selection Criteria

Data management plays an important role in enabling high volume OSS/BSS workloads in real-time environments. To satisfy the increasingly demanding near real-time workload requirements, traditional approaches to data management are being challenged. Service providers find themselves confronted with near real-time analytics in general and data management in particular needs that traditional approaches cannot satisfy. Enterprises now require low latency and high performance capabilities for their ML solutions. Store big, serve small, and understand even smaller…A lot depends on the database selection that initiates and manages the data lifecycle functions. There is a plethora of database solutions capable of satisfying the new expectations. Examples span the full spectrum of traditional RDBMS, OODb, ODB, NewSQL, NoSQL and database-as-a-service offerings.

Data are the essential crafting elements of digital solutions. Queries act as the colored brush, shape defining palettes, implementation algorithms as the master painter capturing autonomous details of the object to be created. Building a satisfied customer requires not only a distinguished brush and talented artist, but also the most appropriate canvas and paint colors. Customer satisfaction, success and loyalty happen when the digital solution is an ERP-as-a-Product, AI-P Ultrasound and Oracle CX. To master these satisfiers, integration is required. Enterprise suite patches and bolt-ons accomplish superficial integration, stitching capabilities using APIs, connectors and definition maps accomplish low-level integration while resident tight framework integration accomplishes magical integration. The higher the level of integration, the better the performance, the attractive the business process, the richer customer experience accomplished.

5.6.2. Data Storage Solutions

Considering the requirements of the solution, we selected four different data stores to support the different application workloads. The systems that were selected consisted of Cassandra, HDFS + Sequence files + Hive, HBase, and RDBMS. As previously mentioned, scalability of the systems was one of our major requirements, supporting our selection of systems, all the systems selected were designed for horizontal scalability. However, eventual consistency was sufficient for two of the data storages – Cassandra for caching and HDFS + Sequence files for historical data, and hence we only needed local consistency for the other two.

In terms of data organization, we used a hybrid design consisting of key-value pairs for Cassandra, and a more loosely defined row-column format for the HBase time series. As previously mentioned, HDFS was only used for large batch processing, and hence simpler formats were sufficient. The data formats used are shown in the table. The logical data partitioning strategy is by workload, with an additional secondary partitioning when necessary in order to achieve uniformity for the data distributions. The secondary data partitions are specified by the collection IDs. The collection IDs can be generated randomly or from a hashing function, and the generated IDs must be included in the data during the different system writes. As mentioned earlier, different data storage systems are important for use cases accessing different data formats. While key-value data stores enable fetching values for specific keys, wide column storage enables vertical fetching, and traditional RDBMS systems are efficient solutions for row accesses. For data records that are not accessed in either of the lazy-access or the vertical access patterns, HDFS provides the best solution.

5.6.3. Real-Time Data Processing Frameworks

Various systems have been developed for processing, to support real-time solutions (or near-real time) as operational stores. The designs of these systems have taken a variety of approaches to provide for low-latency processing. Many of these are designed to work in stream processing modes encompassing data paradigms that differ in concept from traditional batch processing. These frameworks support endless streams of data and provide easy mechanisms to allow splitting of streams into multiple branches to allow for "fan-out" operations to multiple targets. They are built to allow for real-time training of AI models. They support low-latency micro-batching ops; support for Jam, Join, and Aggregate – needed in many use cases; support easy connection to most Pub-Sub Boxes, Message Queues, edge devices, Sensor Networks, or User Interface operations. These frameworks work in either synchronous or asynchronous modes of operations, using pull/push operations. They are designed to allow for building of multi-stage pipelines and replaying of streams in cases of failed operations at downstream targets. They also are designed to allow for scaling-up/down in cases of fluctuating workloads. These frameworks can either provide a high-level programming model or low-level access to define workflows. These frameworks usually have provenance tracking and error-handling capabilities deployed as part of the solution. Given the nature of the above considerations, the types of applications being proposed for supporting either solutions are as follows. However, Petabyte users or very short response-sensitive applications such as certain fraud-detection mechanisms or DDoS prevention mechanisms are examples of types of use-case-sensitive applications that need dedicated code-optimized engines to support the given response-sensitive operations.

5.7. Security Considerations

Security, while necessary for any software architectural design, has especially heightened importance during the design of cloud-based OSS/BSS systems that support critical business functions such as fraud detection, customer billing, top-up recharge, etc. In a traditional, centralized OSS/BSS model, processes such as customer billing, charge calculation for premium services used by customers, or any other customer account related processes are performed locally by a limited number of business-oriented servers, and the actual structured or un-structured data is localized within the data centers of the CSP. With the increasing dependence on cloud services, the OSS and BSS server processes that communicate with cloud-based microservices do so over an external URL in an untrusted Internet environment. Also, the actual structured or un-structured data generated or processed by these critical business functions is often stored in database servers residing at xSP or third-party data centers. Since these OSS/BSS workloads may involve business-critical customer information assets that have been deemed sensitive, it is imperative to ensure that necessary security considerations are factored into the OSS/BSS design process.

In an OSS/BSS integrated offering hosted in the cloud, business consideration tasks such as customer transaction processing, fault management, customer assessment, and other customer-related processes and the actual customer business data are completely offloaded from the business enterprise premises, and are instead hosted in Cloud data center Servers or processed at a third-party vendor who may be managing the integrated OSS/BSS solution for the Communication Service Provider enterprise. Furthermore, the entire implementation of the integrated OSS/BSS solution is owned, managed, and controlled by the third-party vendor who is responsible for delivering business-specific SLAs to the CSP enterprise. Hence, to provide the requisite level of trust, it is imperative that such a third-party vendor solution adheres to the security, disaster recovery, and data protection mechanisms mandated by the CSP to govern such transactions and data transfers.

5.7.1. Data Protection Mechanisms

"Cloud security" is a phrase used very frequently. However, what does it really mean? Cloud Security means protecting the cloud infrastructure, applications, and data from a potential data breach or theft. On-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service are its core characteristics. Cloud service providers maintain only some obligations and responsibilities for achieving this security, while public or private cloud users have to maintain the remaining obligations. Unfortunately, although there is shared responsibility between these parties, it might be difficult to know where the obligations start and end for both parties during sensitive data storage, processing, and transmission on the cloud.

A significant percentage of respondents consider data loss or leakage a concern. Further, many believe insecure interfaces and malicious employees as the next two biggest contributing factors. This highlights the key areas where security mechanisms should be focused. Not only does the sensitive and private data become prone to the risks of a data breach, data loss, and data leakage, but its movement across different locations, application interfaces, and APIs also contributes to the risk. In addition to the hacker malicious attempts, legitimate users who are either service providers or the ones that have access to the cloud or are working inside it, might exploit the plain format of data. The use of unprotected data by these malicious users can cause serious damages. Hence, security mechanisms responsible for protecting data and control interfaces are weighted heavily.

5.7.2. Access Control Models

When applying infrastructure security measures to resource groups in a cloud environment, the cloud architect is faced with the choices of access control models; in particular, choosing between relative open models, such as the entropy of states and entropy of access models, or more conservative models, such as the Bell-LaPadula model. The cloud architect is also faced with the problem of how to factor existing least privilege policies for secure on-premises deployments into new entropic models for access control in a cloud environment. For most resource groups, only primitive policies preventing input parameter modifications are sufficient and more complicated policies are best avoided, at least until full entropic models are available. However, complex policies involving state transitions, inter-model or inter-group transfers, customer administration, and subtree resetting must be made relative open.

A cloud architect has the privilege to set the least privilege policies for the cloud service infrastructure itself, but for some cloud products or customers, additional supervision may be needed. Since the least privilege model gives cloud customers the means to relative close the entropic model complementing the cloud service least privilege model, it is vital that the cloud architect ensure that these customers do not bend or break least-privilege policies of other customers. Critical security best practice dictates that least privilege policies for secure cloud deployments must be implemented as well and must also be relative closed to establish an entitlement model regiment for the cloud environment.

5.7.3. Compliance and Regulatory Challenges

Compliance and regulatory challenges arise from third-party utilities and applications that run on top of the cloud provider core architecture. This set of tools includes Custom Application Development and Applications that utilize the Cloud/Software as a Service model, Third Party Service Integration Tools, Payment and Billing Processing systems and applications. These applications undertake large amounts of financial data processing, especially in the areas of billing and payments, which involves large amounts of credit card and customer financial data processing.

The need for effective and useful Compliance and Regulatory challenges for any use case typically stems from the fact that Cloud Providers and Third-Party SaaS applications and services that leverage the Cloud Provider architecture are governed by a multitude of Compliance and Regulatory stipulations that are different for every country and region.

Some major Compliance and Regulatory stipulations include Financial Stipulations such as Payment Card Industry Foundations and Data Security Standards. Organizations are subject to various directives and regulations regarding the use and processing of personal information. Organizations require compliance with Binding Corporate Rules, Standard Contractual Clauses, or Derogations from regulations, and organizations that are involved in Payment Processing must comply with relevant standards.

Consequent to the varying Compliance and Regulatory requirements, it is important to architect the Cloud Components with the Management of Compliance and Regulatory requirements across the various tools, applications and services that are hosted and/or consume Cloud Service Provider resources. The Plug and Play nature of the tools/services hosted in the Cloud makes it easy for Cloud Computing service providers.

5.8. Monitoring and Performance Optimization

Cloud-based OSS/BSS application often runs critical business functions for service providers, thus imperatively requires constant monitoring and needed resolutions in less than given SLA times. We elaborate on application health checks and monitoring tool options for these types of environments and functions and terminology. For cloud-native applications, we present the pitfalls and best practices to enable auto-optimization and incorporate "design for failure" concepts while architecting the applications.

1. Key Performance Indicators (KPIs)

Measuring point-specific KPIs are essential to monitoring the key cloud-based functions. These functions can be application-specific and platform-specific components. Most of these metrics can be set up and configured using monitoring solutions and services.

2. Monitoring Tools and Techniques

With the advent of cloud and numerous easily deployable solutions including services, the need for simplification and providing intelligent & optimized solution to the consumers has become paramount. The existing and new solutions can be used in such collaborative way that deliver a dependable and cost-effective monitoring solutions to users. Below is a brief overview of available solutions within space.

3. Performance Tuning Best Practices

Tuning of the application functions is as paramount as introducing optimized monitoring/alerting mechanisms during development stages. If the application functions are not well-programmed and if the code has inherent issues, the alert monitors will keep getting triggered often which will take away the focal point of the monitoring system. Proactive and timely tuning of cloud-native applications help mitigate poor user experience and in turn help build and sustain the organizational reputation. Below sections help elaborate the primary and common best practices, but can vary based on cloud provider and configurations.

5.8.1. Key Performance Indicators (KPIs)

Monitoring is a key aspect of any software deployment in a production environment. The complexity and scalability of OSS/BSS workloads deployed on the cloud demand that this aspect receives the utmost attention. Application owners need to ensure that every element in the cloud-based application is appropriately monitored for performance, scalability, and reliability. Data collected from the monitored KPIs over a set period of time can be utilized to create historical trends for not only determining threshold breach alerts for warning of impending issues, but also scaling decisions around when to scale up and scale down the full-stack or elements in the stack. Finally, the monitoring doubles up as a performance tuning tool to determine areas where optimization in usage or cost can be achieved. Monitoring KPIs can be broadly classified into four categories. The first category of KPIs are those pertaining to the cloud infrastructure elements – the underlying physical hardware as well as virtualized images. Using these metrics to create performance dashboards can inform decisions associated with capacity and scaling, and assist in detecting faults and troubleshooting issues associated with the underlying infrastructure elements. All cloud providers provide a set of services to monitor the health of the infrastructure. The second category of KPI metrics are those related to the cloud host system, which is a specialized type of infrastructure element that runs the services that support the application. These metrics provide insights into host issues associated with high usage and resources not available for the application. However, the cloud provider does not provide monitoring services for the host metrics, so it is up to the application owners to install appropriate agent-based tools to capture the metrics. The third category of monitored KPIs are those associated with the containers orchestrating the full-stack services. These metrics assist developers in understanding how well their applications in the containers are performing. Similar to host systems, cloud providers do not have monitoring tools for container metrics installed, so development teams need to instrument suitable tools to capture the metrics.

5.8.2. Monitoring Tools and Techniques

Simply collecting data is not enough. We need to visualize it, be alerted when something is amiss, and we must correlate and analyze the data to see the correlations that point to a potential problem. Monitoring can be accomplished through a combination of many different tools and services, data visualization platforms, third-party tools, custom in-house monitoring systems built specific to each environment, and the vendor tools that are included with the chosen virtualization or cloud platform. When designing your monitoring plan, it is also important to evaluate what we want to accomplish with monitoring, because there are many different degrees of monitoring to support every need. Monitoring provides insight into how an environment is functioning or how a particular application or service is operating. Poor connections or services that time out are good examples of things that monitoring detects. The definition of a particular metric dictates the data that monitoring will present and how it will present it. Alerting is closely tied to monitoring and provides notifications, such as e-mailing or texting, when performance degrades or something exceeds a threshold; for example, when CPU or memory utilization exceeds a defined threshold. There are numerous cloud performance monitoring solutions available today. While not necessarily all-in-one solutions, some of these vendors offer a range of related services that provide insight into multiple metrics. The primary function of cloud performance monitoring is to collect metrics from multi-tiered applications. The second function is to visualize the telemetry data collected from endpoints, cloud services, or across infrastructure systems. Visualization of cloud performance data occurs through interactive dashboards, visualizations like maps, network views, and reports. Cloud performance monitoring is generally agent-based; lightweight agents run on endpoints or across the infrastructure, collecting system metrics from the kernel or hypervisor.

5.8.3. Performance Tuning Best Practices

With experience gained from development and deployment of several important external-facing and internal service-based applications at one of the largest cloud service providers in the world and scaling them to support millions of consumers, in this section, we share our performance tuning best practices used to optimize the performance of the various services and components deploying these applications. Our focus is primarily on best practices for application developers and architects but they are equally applicable for product, service and operations planners. Some key high-level principles for performance tuning are: Records only what you need, at the granularity needed for business function. Human downtime is miniscule compared to machine downtime; optimize for the latter. Caches are your best performance friends but ensure that they are validated correctly and are updated properly or you may introduce inconsistencies. If a synchronous call is taking too much time to process, consider if it should be Asynchronous; perhaps introduce an A/B function. Don't forget to profile for non-responsiveness; it could be that your code, network or storage is affecting overall performance. Use the simplest possible solutions for your services.

Application architecture and frameworks have unique characteristics that define their performance tuning considerations. Programming language, runtime libraries and frameworks, databases, web servers, messaging systems can all have their respective tuning parameters and will affect the performance of services built by developer teams. For the purposes of this document, we choose to limit context-specific details on performance tuning of these systems and parameters to avoid overwhelming product developers and architects. Before delving into more details from our experience with these applications, we summarize common considerations for general SLAs and OLTP workloads.

5.9. Conclusion

Although cloud infrastructure has been a major technology evolution over traditional on-premise solutions, and cloud development continues to drive innovation in the Telecommunications industry, these advancements have not yet been able to deliver on promises of a real-time, scalable OSS/BSS architecture to support subscriber growth and engagement. The Telecommunications industry is hitting the limitations of Cloud Scalability, Performance and Costs, and discussing the possibility of abandoning Cloud and Microservices flexibility to go back to monolithic, on-prem systems. Traditional Telecommunications OSS/BSS workloads, performing internal-to-the-TELCO-system transfers at sub-second latencies, on data volumes that can only be managed at a Terabyte scale, are forcing the Telecommunications SP architecture to rethink these discussions. Real-time management of communications systems may push the OSS/BSS components to move outside of Cloud Infrastructure – where Latency is a Design Parameter. We believe that Current Workload Trends – based on Growth - are going to push Operations Workloads in a different direction than Development Workloads.

The OSS/BSS Architecture must handle several conflicting characteristics, scalability of data and services over time (Management Growth, Subscriber Growth, Revenue Growth), the move to a high-volume, high-sustained performance fluctuating workload with unpredictable spikes and dips, low transaction latencies while performing trillion-object transactions, support extreme all Business and IT Change Rates asymmetry between Real-Time Event Notifications and Subscriber Data Value-generation consistency, with Part of the Business Model Handling Frequent, Impulsive, Limited Time Offers. Managing THIS SYSTEM with an OSS/BSS that uses the same Cloud Infrastructure and Share's Cloud Architecture Characteristics driving Business Change Rate becomes a Fail-Too-Often Proposition. You need an Architecting Approach with an Architecture Management that Can Support Latency for Control Workloads with Continuous Architectural Evolution.

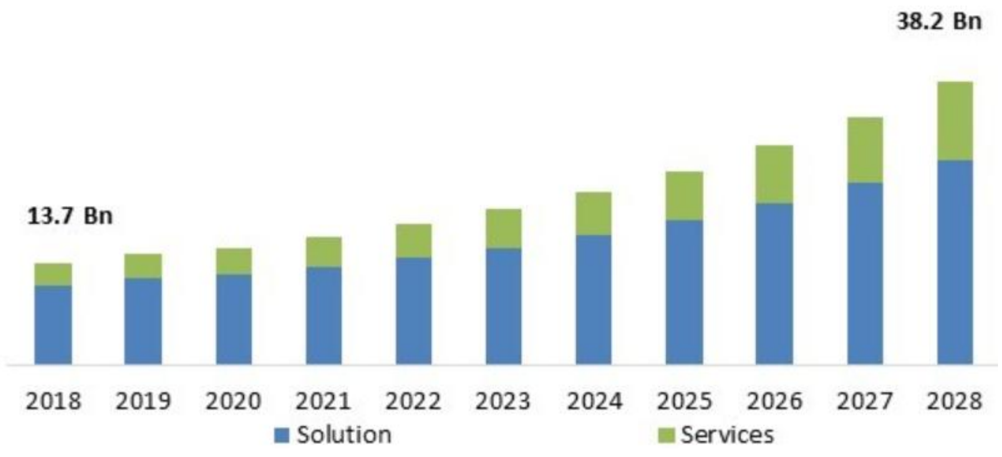

Fig 5 . 3 : Cloud OSS BSS Market Size

5.9.1. Summary and Future Directions

While the applications and services of the telecom industry are being offered at a rapid pace to consumers and businesses, the telecom infrastructure to support these services is evolving at a slower pace to satisfy the demands of innovation in this marketplace. The Telecom OSS/BSS application landscape is not well suited to support this rapid deployment of telecom applications. The primary objective of our work is to address some of the challenges posed by this challenge. Telecommunication service providers are rethinking their OSS/BSS workloads design principles and moving towards a service-oriented architecture. OSS/BSS workloads are being decoupled from monolithic code bases and both legacy workflows and its associated data are being repackaged into microservices and micro-microservices, governed by standardized APIs and opened for sharing. A growing list of additional objectives are being incorporated in this rethinking, ranging from requirements of multi-tenancy to dynamic scalability to lower total cost of ownership.

In this work, we conduct a detailed exploration of the architectural implications of supporting these objectives in specific telecom OSS/BSS workloads when deployed in a public cloud environment. We present a detailed description of four OSS/BSS workloads addressing commercial, network, inventory, and trouble management. These workloads, which have been jointly optimized for satisfying non-functional requirements, also demonstrate the bearing of data on overall workload performance. Our results indicate that potential improvements in infrastructure optimization software has still a major role to play in lowering the total cost of ownership of OSS/BSS workloads. In conclusion, we present a unified cloud end-to-end design principle for Concurrent Engineering Services. We expect and anticipate that our research will inform and accelerate the ongoing move to a cloud service-oriented architecture across telecommunications service providers. We hope that our excerpts will inspire further research on other telecom workloads and further development of the architecture to support end-to-end design principles for other service domains.

References

Khajeh-Hosseini A., Greenwood D., Sommerville I. (2010). The Cloud Adoption Toolkit: Supporting Cloud Adoption Decisions in the Enterprise. Software: Practice and Experience, 42(4), 447–465. https://doi.org/10.1002/spe.1072

Andrikopoulos V., Song Z., Strauch S. (2014). Cloud Instance Management and Resource Prediction: A Performance Study. Journal of Cloud Computing, 3(1), 1–15. https://doi.org/10.1186/s13677-014-0021-3

Botta A., De Donato W., Persico V., Pescapé A. (2016). Integration of Cloud Computing and Internet of Things: A Survey. Future Generation Computer Systems, 56, 684–700. https://doi.org/10.1016/j.future.2015.09.021

Ghosh R., Naik V.K. (2016). BSS/OSS Transformation for Telcos in a Virtualized Environment. Bell Labs Technical Journal, 20(1), 161–177. https://doi.org/10.15325/BLTJ.2015.2459734

Erl T., Khattak W., Buhler P. (2017). Cloud Computing Design Patterns. Prentice Hall, 368–410. https://doi.org/10.5555/3107705