IOWN GLOBAL FORUM™

Data Hub Functional Architecture

Classification: APPROVED REFERENCE DOCUMENT

Confidentiality: PUBLIC

Version 2.0

July 20, 2023

[IDH]

Legal

Contents

List of Figures

List of Tables

1. Introduction

The IOWN Global Forum's Data Hub (IDH) is a data management and sharing solution that enables fast and trusted data processing, usage, and exchange among multiple parties or locations.

The actual deployment of the IDH solution is assumed to be based on application functional nodes that are provisioned by the IOWN Global Forum Data-Centric Infrastructure (IOWN DCI), interconnected through the IOWN Global Forum Open All-Photonic Network (IOWN Open APN), and protected with the IOWN Global Forum Security for higher performance, better quality, and more robust security. To accommodate various application requirements, the actual implementation of the IDH solution will take multiple forms, called IDH services. Later in this document, these service are grouped into service classes so that discussions around IDH service requirements and implementation models can be streamlined. This document is designed to be a well-organized guide for service providers who develop IDH services.

Each IDH services will provide various APIs so that the IDH service and data managed within the IDH service can be easily shared across multiple applications and parties. Some of these APIs will be designed to be functionally compatible with well-adopted data management services, such as Apache Kafka/Pulsar, AWS S3, and SQL so that IDH service usage and development of applications that leverage IDH services can be facilitated. In addition, to prevent both internal and external attacks and achieve trusted data exchange among multiple parties while maintaining data ownership, the IDH provides unified mechanisms for protecting data, securing data access, and controlling data usage.

The critical requirements for driving the development of such IDH services, today's technology gaps, the desired functional architecture of IDH services, and practical adoption scenarios of IDH services for various IOWN Global Forum use cases are described in subsequent chapters of this document.

2. Use Case Requirements and the Necessity of the IOWN Global Forum Data Hub Technology

In the IOWN Global Forum, various use cases are envisioned as described in the IOWN Global Forum Cyber-Physical System Use Case document [IOWN GF CPS UC] and the IOWN Global Forum AI-Integrated Communications Use Case document [IOWN GF AIC]. To realize these use cases as viable business and technical services, an enormous amount of diverse data types needs to be collected, processed, used, and shared effectively, efficiently and in a timely manner. Table 1 below shows the critical requirements expected for these use cases.

Table 2-1, IOWN Global Forum Use Cases and High-level Data Requirements

Use Case	**Expected Requirements**			
	Data Variety	Data producers and consumers	Data Collection Bandwidth	Producer-Consumer Latency

CPS:
Area Mgmt. - Security:
Immediate alerting security officers and possibly other near-by people, when any dangerous situation is detected by collecting and analyzing surveillance video and other sensor data.
	• Video cameras
	• LiDAR sensors
		Data Producers
		• Video cameras:
		  ○ 1,000 per monitored area (e.g., building) x 100 - 10,000 monitored areas
		  ○ Continuously sending data
		• LiDAR Sensors:
		  ○ 1,000 per monitored area (e.g., building) x 100 - 10,000 monitored areas
		  ○ Continuously sending data
		Data Consumers
		• Security officers:
		  ○ Up to 1,000 per monitored area
		  ○ Receiving alerts when detected
		• Security monitors:
		  ○ Up to 50 per monitored area
		  ○ Displaying the selected cameras
			• 45-60 Mbps per camera (assuming Full HD motion JPEG)
			• 600-800 cameras i.e., 27-48 Gbps per building
			• Thousands or millions of buildings (depending upon the area size)
				• From data collection through analysis to alerting:
				1 sec, ideally 100 milliseconds

CPS:
Area Mgmt. -
Disaster Notification (e.g., earthquakes, tsunamis, floods, wildfires):
Immediate alerting people and relevant systems in the area when any disaster of concern is detected by collecting and analyzing various sensor data.
	• Sensor data
	• Video Cameras
	• Citizen data (Some snapshot data)
	• Other relevant data such as terrain data
		Data Producers
		• Various Sensors
		  ○ 100,000 per square km
		  ○ Continuously sending data
		Data Consumers
		• Various user devices (e.g., smartphones of residents)
		  ○ Hundreds of millions
		  ○ Receiving alerts when detected
			• 0.1-1 MB every second or 10 seconds per sensor	• From data collection through analysis to disaster detection: Sub-second

CPS:
Mobility Mgmt. -
Safety Maneuver:
Remotely assisting vehicle operation by detecting dangerous situations based on data collected from the vehicle of concern.
	• Vehicles' cameras
	• Vehicles' LiDAR sensors
	• Road-side sensors
		Data Producers
		• Vehicles equipped with various external sensors
		  ○ 120 per per 1 km road segment
		  ○ Continuously sending data when they are running
		• Road-side sensors, such as camera
		  ○ 10 per 1 km
		  ○ Continuously sending data
		Data Consumers
		• Vehicles which may not be equipped with external sensors
		  ○ More than 120 per 1 km road segment
		  ○ Receiving safety maneuver signals when required
			• Up to 1 Gbps per vehicle
			• Up to 12 Gbps per cell (A cell corresponds to 100m road segment)
			• tens of millions of vehicles
				• From data collection through analysis to feedback controls: 10 - 500 milliseconds (depending upon the required control)

CPS:
Mobility Mgmt. -
Energy Optimal Routing:
Estimating traffic congestion, weather, etc. by collecting data from running vehicles, and providing guidance on the best route to running vehicles.
	• Vehicles' cameras
	• Vehicles' LiDAR sensors
		Data Producers
		• Vehicles equipped with various external sensors
		  ○ 120 per 1 km road segment
		  ○ Continuously sending data
		Data Consumers
		• Vehicles which may not be equipped with external sensors
		  ○ More than 120 per 1 km road segment
		  ○ Receiving optimal routing information when required/requested
			• Up to 1 Gbps per vehicle
			• Up to 12 Gbps per cell (A cell corresponds to 100m road segment)
			• Up to 1200 Gbps per service area
				• From data collection through analysis to local traffic situation knowledge updates:
				10 seconds, ideally 1 second

CPS:
Industry Mgmt. -
Factory Remote Operation:
Real-time display of the operating status of factory equipment, which includes images and analyzed supplementary information, to realize remote operation.
	• High-definition (4K/8K) cameras	Data Producers
		• Machines and/or robots
		  ○ Up to 10,000 per factory
		  ○ Continuously sending data when they are in operation
		Data Consumers
		• Monitors for remote operation
		  ○ Up to hundreds per factory
		  ○ Displaying the status of selected machines and/or robots
			• 10 Gbps per plant	• From data collection through analysis to display:
				Sub-second to a second

CPS:
Industry Mgmt. -
Process Plant Automation:
Remotely controlling factory machines and drones by collecting and analyzing sensor data from the machines and drones of concern.
	• Robot/Drone's High-definition (8K) cameras
	• Robot/Drone's Sound/Odor sensors
		Data Producers
		• Machines and/or robots
		  ○ Up to 10,000 per factory
		  ○ Continuously sending data when they are in operation
		Data Consumers
		• Machines and/or robots
		  ○ Up to 10,000 per factory
		  ○ Receiving control signals when required
		• Factory Engineers
		  ○ Up to thousands
		  ○ Receiving alert signals and other information when required
			• 2.35 Gbps per robot/drone	• From data collection through analysis to feedback controls:
				10 milliseconds, Sub-second, or a second (depending upon a use case scenario assumed.)

CPS:
Network Infrastructure Mgmt. -
Network Device Failure Prediction:
Detecting and alerting network device failures by collecting and analyzing network telemetry data from them.
	• Network telemetry data
	• Non-network data, such as temperature
		Data Producers
		• Network devices
		  ○ 100,000 (as a scale of the initial deployment.)
		  ○ Continuously sending data when they are in operation
		Data Consumers
		• Relevant applications, e.g., network monitoring system
		  ○ A few to tens
		  ○ Receiving alert signals when detected
			• 800 Mbps in total for a network system with 100,000 network devices, assuming:
			  ○ 100 metrics are managed for each network device on average
			  ○ Each metric generates 100 bytes every 10 second on average
				• From data collection through analysis to notification:
				10 seconds

CPS:
Network Infrastructure Mgmt. -
Telegraph Pole Collapse Detection:
Detecting telegraph pole collapses by collecting and analyzing network telemetry data from them, and notifying it to several applications.
	• Optical fiber sensor	Data Producers
		• 100,000 (as a scale of the initial deployment.)
		• Continuously sending data when they are in operation
		Data Consumers
		• Relevant applications, e.g., network monitoring system
		  ○ A few to tens
		  ○ Receiving alert signals when detected
			• 32 Gbps per fiber sensing interrogator	• From data collection to through analysis to notification:
				1 - 10 seconds

CPS:
Healthcare Mgmt. -
Disease Outbreak Prediction:
Detecting disease outbreak by collecting and analyzing various sensor data, and notifying it to several applications.
	• Environmental sensor (air quality, etc.)
	• Wearable sensor (vital signs, etc.)
	• Additional data (such as, symptoms of cough, etc., provided from crowd sources.)
		Data Producers
		• Various sensors
		  ○ 2,000 per 1 km^2
		  ○ Continuously sending data
		Data Consumers
		• Relevant applications, e.g., disease outbreak monitoring system
		  ○ A few to tens
		  ○ Receiving early warning signals of disease outbreak when detected
			• 1 Gbps per 25km^2 cell	• From data collection through analysis to detection:
				Minutes to hours

CPS:
Smart Grid Mgmt. -
Renewable Energy Flow Optimization:
Enabling automated operation of the power grid system that relies on renewable energy as the main power supply by controlling power storage, discharge, and distribution.
	• Sensor data from prosumer
	• External data sources (weather data, human flow data, etc.)
		Data Producers
		• Prosumer devices (e.g., smart meters, electric vehicles, etc.) and power grid system equipment (e.g., switchgear, transformer, etc.)
		  ○ More than 100,000
		  ○ Continuously sending data
		Data Consumers
		• Prosumer devices and power grid system equipment
		  ○ More than 100,000
		  ○ Receiving control signals when required
			• 24 Tbps in total for sensor data from prosumers
			• 128 Pbps for electric vehicles in 2030
			• 3.2 Gbps for other data sources
				• From data collection to display:
				50 milliseconds
				• From data collection through analysis to feedback controls:
				200 milliseconds

CPS:
Society Mgmt. -
Sustainable Society:
Solving various issues for realizing a sustainable society by collecting and analyzing various data relevant to human activities.
	• Various big data sources for understanding society	Data Producers
		• Various sensors
		  ○ 400 millions (assuming sensors are installed every 5 m^2 to cover 2,000 km square city.)
		  ○ Continuously sending data
		Data Consumers
		• Relevant applications, e.g., transportation optimization system
		  ○ Tens to hundreds
		  ○ Receiving updated social insights periodically or when required
			• 10 PB in total across various data sources	• From data collection through analysis to report:
				Hours to days

AIC:
Entertainment - Interactive Live Music:
Building a virtual space in which many audiences enjoy music together through avatars.
	• Video camera of audiences & artist(s)	Data Producers
		• Audience and artist devices
		  ○ Up to 1 million
		  ○ Continuously sending motion, audio, and other data when connected
		Data Consumers
		• Audience and artist devices
		  ○ Up to 1 million
		  ○ Continuously receiving image and audio data when connected
			• From artist(s)
			  ○ 56.35 - 112.74 Gbps
			• From audiences
			  ○ 7.23 Gbps per audience
				• Motion-to-Photon: 10 milliseconds
				• Time-to-present: 70 milliseconds

AIC:
Entertainment - Interactive Live Sports:
Providing a sports viewing service that radically changes the experience value by reproducing scoring scenes from the player's point of view, etc.
	• Video camera capturing a sport event	Data Producers
		• Player devices
		  ○ A few to tens
		  ○ Continuously sending motion, audio, and other data when playing
		Data Consumers
		• Player devices
		  ○ A few to tens
		  ○ Continuously receiving image and audio data when connected
		• Audience devices
		  ○ Up to 100,000
		  ○ Continuously receiving image and audio data when connected
			• 14-230 Gbps for video input streams	• Motion-to-Photon: 10 milliseconds
				• Time-to-present: 70 milliseconds

AIC:
Entertainment - Cloud Gaming:
Providing high-spec game services that allow so many such as 100,000 players to participate and compete simultaneously using low-cost devices.
	• Game operation input data from the game users	Data Producers
		• Game player devices
		  ○ Up to 100,000
		  ○ Continuously sending operation, audio, and other data when playing
		Data Consumers
		• Game player devices
		  ○ Up to 100,000
		  ○ Continuously receiving image and audio data when playing
		• Audience devices
		  ○ Up to 100,000
		  ○ Continuously receiving image and audio data when watching
			• 10 Kbps per console *assuming collecting 100-bit signals every 10 msec	• Motion-to-Photon:
				• 10 - 600 milliseconds

AIC: **Entertainment - On-the-Go Entertainment:** **Providing the same entertainment services described above to users in a running vehicle through the mobile network**	Same as other entertainment services.	Same as other entertainment services.	Same as other entertainment services	Same as other entertainment services despite the usage of mobile network

AIC:
Remote Operation - Professional Learning:
Providing a service to learn professional works through experience in a virtual space.
	• Operation input data from students	Data Producers
		• Teachers and students devices
		  ○ Up to 100 per lesson
		  ○ Continuously sending motion, audio, and other data during a lesson
		Data Consumers
		• Teachers and students devices
		  ○ Up to 100 per lesson
		  ○ Continuously receiving image and audio data during a lesson
			• 100 Mbps per console *assuming collecting 10-Kbit signals every 0.1 millisecond	• Time-to-control: sub-millisecond - 20 millisecond

AIC:
Navigation - Ultra XR Navigation:
Providing an XR navigation service that allows its users to experience the target area as if they were there
	• Input data for XR device	Data Producers
		• Client applications (e.g., web browsers) of XR Content Service Providers
		  ○ Tens of thousands
		  ○ Uploading XR contents adhoc-ly
		Data Consumers
		• Ultra XR Navigation service user devices (e.g., smartphones)
		  ○ Millions
		  ○ Receiving XR navigation information when using the Ultra XR Navigation service
			• 120 Mbps per XR device
			• 100,000 XR devices in total in a service area
				• Time-to-present: 16 - 33 milliseconds

AIC:
Human Augmentation - Mind-to-Mind Communications:
Providing a service that enables smooth communication between people by taking into consideration speakers' feelings, cultural background, etc., and adding supplementary information.
	• User's communication data	Data Producers
		• User devices
		  ○ More than 10 millions concurrently
		  ○ Continuously sending video and audio data during a call
		Data Consumers
		• User devices
		  ○ More than 10 millions concurrently
		  ○ Continuously receiving video, audio, and other supplementary data during a call
			• Up to 290 Gbps per user	• Time-to-present for auditory sensing: the order of 100 milliseconds
				• Time-to-present for visual sensing: the order of 10 milliseconds

AIC:
Human Augmentation - Another Me:
Instantiating "another me" by converting his/her knowledge and experience into data, and modeling his/her judgment criteria and preferences, and letting "another me" interact with others to learn more from it.
	Depending up on scenario.
	(For instance, if talking with an "Another Me" instance by voice, voice data is the input
		Data Producers
		• Service user client applications
		  ○ Up to millions
		  ○ Registering/updating "another me" model data at an adequate time
		Data Consumers
		• Service user client applications
		  ○ Up to millions
		  ○ Receiving answers from "another me" contacted
		• Another me owner client applications
		  ○ Up to millions
		  ○ Receiving interaction history/statistics and lessons learned
			Depending up on scenario.	• Time-to-present for auditory sensing: the order of 100 milliseconds
				• Time-to-present for visual sensing: the order of 10 milliseconds

As clearly seen in the above matrix, the future use cases envisioned by the IOWN Global Forum must meet diverse and stringent data processing and sharing requirements. Therefore, the development of new technologies is absolutely necessary.

For example, to meet real-time requirements, it will be required to process data closer to its origin. Also, to cope with a large amount of data flow, it will be necessary to have a drastically improved distributed data management engine that runs on an ultra-high-speed and high-quality network. And finally, it will be necessary to develop robust data security and data usage control mechanisms, to enable secure data re-usage across multiple parties.

Based on these requirements and goals, this document first organizes the existing data management technologies in Chapter 3, analyzes the gaps in the existing technologies against the IOWN Global Forum requirements in Chapter 4, and determines new implementation models to fulfill the gap in Chapter 5. In addition, this document discusses how various use cases can be realized with the support of the IDH services in Chapter 6.

3. IDH Service classes

Given the variety of requirements from different use cases described in Chapter 2, it would not be wise to have just one IDH implementation model with a corresponding API set, because it results in the necessity of developing technology that satisfies extreme requirements on all fronts, but only a certain percentage of its capabilities is used depending on the use case, leading to wasted resources and increased costs. With this regard, various IDH implementation models will need to be developed and used.

In this chapter, typical patterns of such implementation models, called IDH service classes, each of which represents a template that is referred to when a service provider implements a corresponding service, are outlined. In addition, when describing these IDH service classes, they are categorized into either a basic service class that provides basic services for a specific data variety or an applied service class that provides composite services and/or manages multiple data varieties.

3.1. Basic Service Classes

Basic service classes are a set of fundamental IDH services, which are inevitable in digital experience (DX) services. Each basic service class is designed so that data in a specific format can be accessed, queried, analyzed, and/or queued, in an efficient and secure manner.

The outline of such basic service classes are described:

- Distributed Relational Database
  - Data format: Tabular
  - Objectives: To accelerate reads and writes of data.
  - Design: Storing data in predefined tables and generating indexes in advance for frequently referenced columns. Strict transaction consistency is treated as default but can be weakened.
- Key-Value Store
  - Data format: A collection of key-value pairs
  - Objectives: To manage various data flexibly and with high scalability.
  - Design: Simply storing various data as a “Value”, each of which can be identified by “Key”. Transaction consistency is generally not considered, but it can be configured to ensure some level of consistency.
- Graph Store
  - Data format: Link-centric
  - Objectives: To efficiently manage and explore a wide variety of relationships.
  - Design: Storing related links together for each of target entities, so that exploration can be accelerated.
- Message Broker
  - Data format: Messages (Files) and relevant metadata (properties) with partitioning and sequence numbers.
  - Objectives: To build a pipeline to support fast insertion and retrieval of messages.
  - Design: Storing messages in the order in which they were inserted.
- Distributed File Storage
  - Data format: Files and directories

  - Objectives: To organize a large number of files hierarchically to support continuous exploration and access.
  - Design: Grouping files based on directory hierarchy for performance, storing file groups in a distributed manner by using multiple servers for scalability, managing directory hierarchies and metadata in a database, and placing the frontend servers so that file system access can be mounted at their clients.
- Object Storage
  - Data format: Objects (Files) and URI
  - Objectives: To store a massive number of objects and support ad-hoc access
  - Design: Storing objects, or chunks of objects when object is so large, in a distributed manner by using multiple servers for scalability, managing their metadata in a database, and placing the frontend servers so that REST-based object access can be provided.

Please refer to Annex A for details of these basic service classes.

3.2. Applied Service Classes

An applied service class represents a template of an enhanced/extended IDH service, which may inherit two or more other IDH service classes. By providing multiple composite functions and/or managing different varieties of data together in one place, services built on any applied service class definition can reduce the necessity of data movement and process data at higher density, which is considered inevitable for realizing real-time services and reducing energy consumption, i.e., achieving the goals of IOWN Global Forum's solution development.

Given the positioning of such applied service classes, it is expected that a large number of service classes may actually be developed. Below is an example of such applied service classes:

- Context Broker
  - Inherited Service Classes: Message Broker, Distributed RDB or KVS, and Graph Store
  - Data format: Messages (Files) and relevant metadata (properties) with partitioning and sequence numbers.
  - Objectives: To accelerate data distribution processing faster than the message broker. When implementing IoT systems with today's message brokers that require different processing depending on data conditions, consumers of the message broker need to parse data to determine which node to forward the data for subsequent processing such as generating data linkage. By adding query and data linkage functions, the necessity of having such a mechanism is eliminated, which reduces the data processing latency and the energy consumption.
  - Design: Adding RDB-like or KVS-like query engine that parses properties of messages on top of the Message Broker, and also adding a Graph-based data linkage function at the output layer of the Message Broker.
- Converged DB
  - Inherited Service Classes: Distributed RDM, KVS, Graph Store, and Message Broker
  - Data format: Tabular, a collection of key-value pairs, and Link-centric
  - Objectives: To manage, update and use varieties of data in one place in a consistent manner. This is an essential requirement as use cases like the Digital Twin need to work with different data varieties simultaneously. Without this mechanism, data would need to be moved among various IDH services and application nodes for this purpose, which should be avoided for the realization of real-time services.

    - Design: Instantiating multiple database mechanisms in one place. Then, adding an intelligent request handling layer that distributes client requests to suitable internal query engine according to the data structure requested and does data conversion as required.

- Virtual Data Lake (Federated Storage)
    - Inherited Service Classes: Distributed File Storage and Object Storage
    - Data format: Files-directories and Objects-URIs
    - Objectives: To streamline data sharing across multiple parties. With any current file storage and/or object storage service, efficient sharing of data among multiple parties is not possible because of security risks and a lack of integrated data access. A new technology that enables faster and more secure data sharing needs to be developed.
    - Design: Integrating multiple Distributed File Storage services and Object Storage services. To provide uniform access across multiple storage services, a global namespace is constructed, directories and URIs are merged, and filenames and object names are merged.
- Virtual Data Lake House
    - Inherited Service Classes: Distributed Relational Database, Key-Value Store, Graph Store, Message Broker, Distributed File Storage, and Object Storage
    - Data format: Tabular, a collection of key-value pairs, Link-centric, Files, and Objects-URIs
    - Objectives: To unify and streamline data access across various IDH services. With today's implementation model, a type of data virtualization technology, which provides a virtualized view across multiple data services, is too slow, and many applications need to connect to each database separately for performance. This situation needs to be improved.
    - Design: Unifying APIs, and adding federation mechanism between IDH services, so that different types of data stored in different IDH services can be accessed together. Regarding APIs, SQL will be enriched so that key-value pairs, graph, and semi-structured data such as JSON can be accessed jointly.

Please refer to Annex A for details of these applied service classes.

4. Gap Analysis on Today’s IDH-like services

Many relevant technologies corresponding to the Data Hub classes in the previous chapter already exist in the market. However, to realize the IOWN Global Forum use cases, there are still so many gaps that need to be fulfilled. In this section, origins of such gaps in today's technology landscape and the resulting implementation styles that are compelled to be used for building IDH-like services are explained.

4.1. Origins of Gaps

There are some fundamental issues with today's network and data center technologies when building IDH services to meet the requirements of the IOWN Global Forum use cases described in Chapter 2, which are described below:

- Inter-DC Network Quality

  Inter-datacenter network quality is not so good, i.e., the possibility of packet reordering and loss is even higher in today’s cloud. Over such networks, TCP is used more frequently than other protocols such as UDP to ensure reliability, which is a considerable burden. And this will worsen the performance of workloads such as geo-distributed DBs and storage systems.

- Intra-DC Network Quality

  In today's cloud-like hyper-scale data center, the network quality connecting resources is not good enough. For example, 1-way latency can exceed 1 ms, and packet loss can exceed 1% during periods of severe network congestion. Bandwidth is similarly unstable. This is basically due to over-subscription, where the total bandwidth allocated to each resource cannot be supported by the horizontal network connecting the resources in a data center.

- Storage Performance

  Performance of the storage system is not enough, too. This is because storage is usually designed to keep synchronous copies of data on multiple devices in order to ensure data persistence, however, with above network characteristics, it is not a small burden and destabilizes performance. As a result, an intra-DC storage system connected through a storage area network in today’s cloud tends to run slower than the direct-attached storage device, for example, the response time of the storage system goes beyond a few tens of milliseconds sometimes, which is not good for database-type of workloads. Also, geo-redundant storage systems are rarely used.

- Data Sharing Performance

  When data is jointly processed by multiple servers, a certain amount of data exchange is required between servers, for such as redistribution of data for shuffling, which greatly affects overall performance. To accelerate such data sharing, some implementations use RDMA Fabric to connect servers. However, applicability of RDMA in a production environment tends to be limited to local clusters of limited size due to difficulty in meeting no packet loss and reordering requirements as well as due to the performance degradation proportional to distance caused by the pause signal, and it is only used for very limited purposes in today’s cloud environments.

- Accelerator Usage

  Accelerators such as GPUs and FPGAs have the potential to achieve more than ten times better performance, cost efficiency, and energy efficiency than software-based processing on general-purpose CPUs. However, such accelerator usage is limited due to the necessity of local installation on each server, and the lack of direct exposure to external clients to streamline data loading. Thus, with today's networking and data center technologies, it is not allowed to share accelerator resources across multiple general-purpose compute resources at high speed and low overhead.

Figure 4-1-1 below shows an image of these bottleneck points.

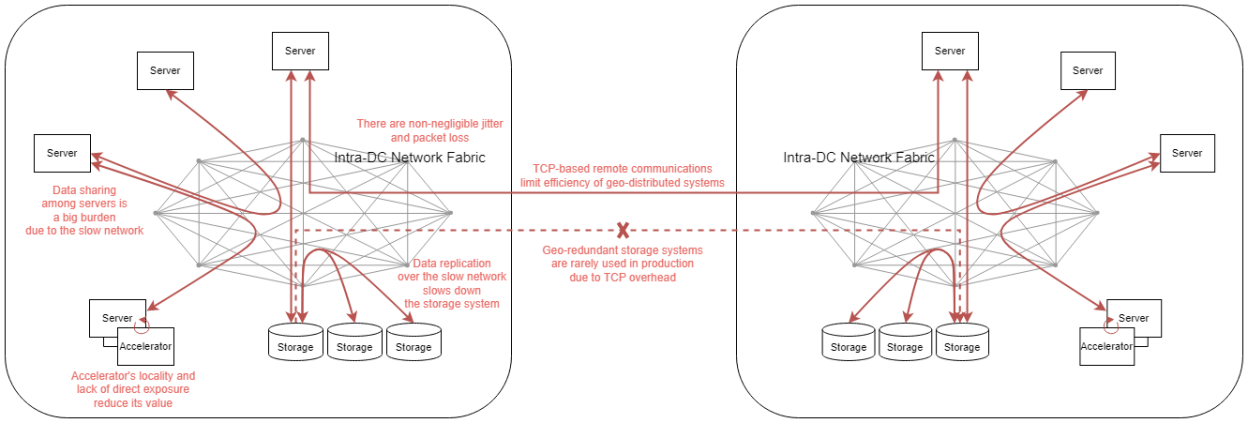

Figure 4-1-1, Origins of Gaps in today's technology landscape

4.2. Today's Implementation Models and Inherent Gaps

Due to the existence of the issues described in section 4.1, some trade-offs have been accepted when designing an implementation model of today's IDH-like services, which are described below:

- Completion of data processing within one place

  Not good Inter-DC Network Quality hinders the implementation of data services utilizing distributed data centers. This is because the processing efficiency is greatly reduced when doing so. As such, today's IDH-like service implementations are only designed to be deployed within a single data center or, at best, in a cloud region, i.e., a set of nearby availability zones of today's cloud infrastructure. In fact, today's cloud is a collection of such services. Therefore, the scale of the cloud data center becomes huge, and the number of available cloud regions is very limited, e.g., about 30 globally at most. This means that all data must be transferred to one of the cloud data centers no matter where it is generated and all data processing must be completed there. This is wasteful in terms of energy consumption and definitely not the best solution.

- Asynchronous data replication as a foundation to build a scalable transactional Distributed RDB

  This design goal is to build a scalable transactional RDB system in today's technology landscape by having read-only replicated data close to each RDB server. However, such replication tends to be conducted asynchronously due to slow networking performance and data consistency cannot be guaranteed. Also, to reduce the data replication burden over the slow network, only transaction or change logs may be propagated. In that case, the performance becomes better, although the table data must be rebuilt from log data at each server, which leads to an increase in cost and energy consumption.

- Sharded data processing to build a scalable KVS, Message Broker, and/or analytical Distributed RDB

  This design is used to build a scalable system such as KVS, without using the slow remote storage system. Relevant data may be grouped together and placed in the same server so that inter-server join operations can be minimized. However, the data shuffling process (data rearrangement process) and inter-server communication are still not zero. Rather, such processing may even become more important for advanced DX services that need to handle data from various angles. In addition, sharded systems also constrain dynamic scalability. This is because even with algorithms such as consistent hashing described in Annex B, there needs to be a certain amount of data movement during scale-out and scale-in operations. Therefore, IDH-like services in today's Cloud tend to have limited online scalability.

- Usage of a cache layer

The slow network and the slow storage system make synchronous and shared processing difficult and result in heavy use of asynchronous and distributed data processing. This means that the data will be distributed and managed in each process. In fact, today's cloud offers a variety of cache data management services to streamline such implementations. However, such an architecture increases end-to-end latency and causes similar data to be copied multiple times, which is undesirable in terms of cost and energy consumption.

- Accelerator in use only in a laboratory

  It is conceivable to speed up data processing such as queries in a Distributed RDB by using accelerators such as GPU and FPGA. In fact, such open source software, or OSS, has been developed and tested. However, the reality is that doing so does little to improve performance and cost efficiency, and often makes them worse. This is due to the necessity to install the accelerator on each server, which may not be required all the time, and the necessity to preload the data to the accelerator.

Figure 4-2-1 below shows images of such designs used today.

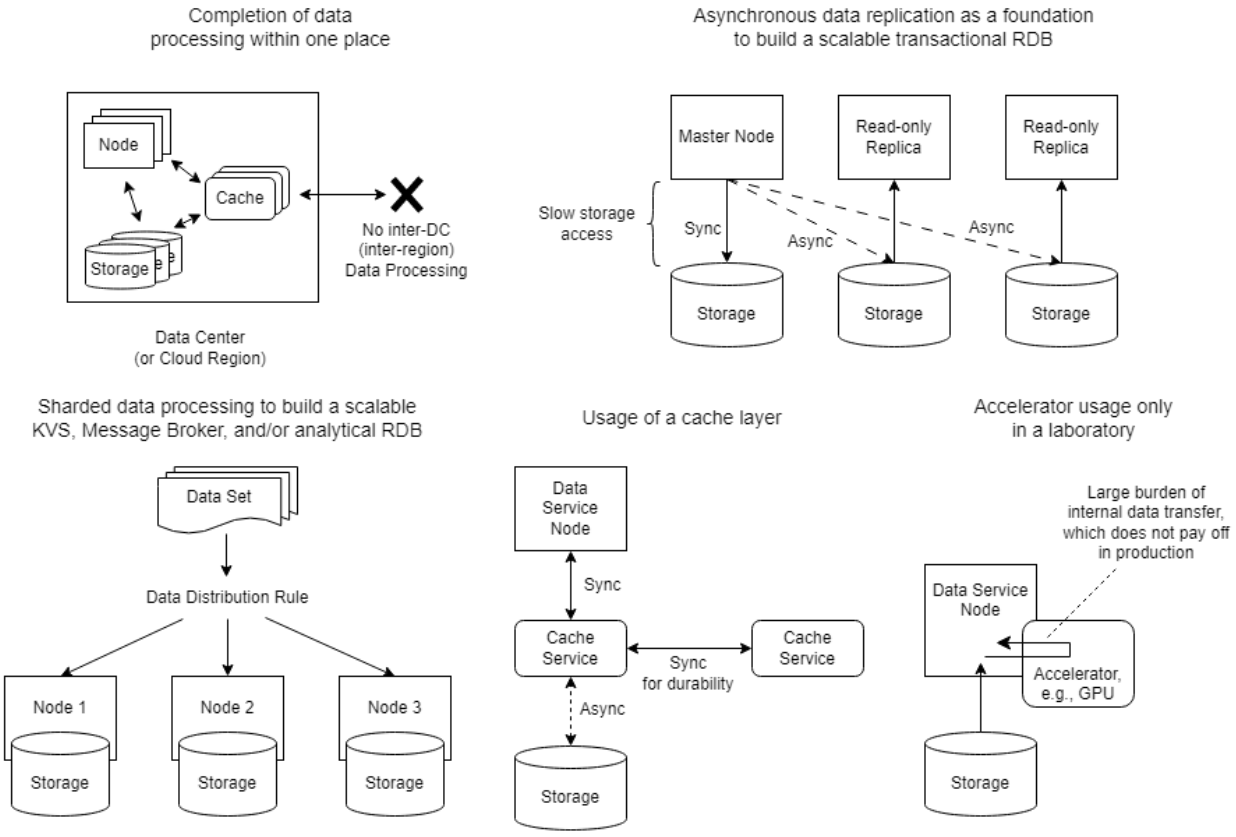

Figure 4-2-1, Today's Implementation Model Styles of IDH-like services

Please refer to Annex B for more detailed implementation models of IDH-like services today and their gaps.

5. IDH Functional Architecture and Acceleration Mechanisms

This section is to explain a couple of possible IDH functional architectures and acceleration mechanisms that are expected to solve the gaps described in Chapter 4.

5.1. Generic Architecture Structure

Regarding the overall structure of IDH, a 4-tier model shown in Figure 5-1-1 is assumed so that geographically and functionally distributed data management, processing, and sharing can be supported and gaps described in Chapter 4 are well addressed. When deploying this model, the client tier can be located in any customer premises, such as a smart factory, or a smart building, and the frontend tier can be located in a geographically separate data center from the data services tier. The data service tier and the storage tier are often located in the same data center, but it is not denied that they are located somewhat apart. Such distributed data processing and management is realized by the IOWN Global Forum OpenAPN.

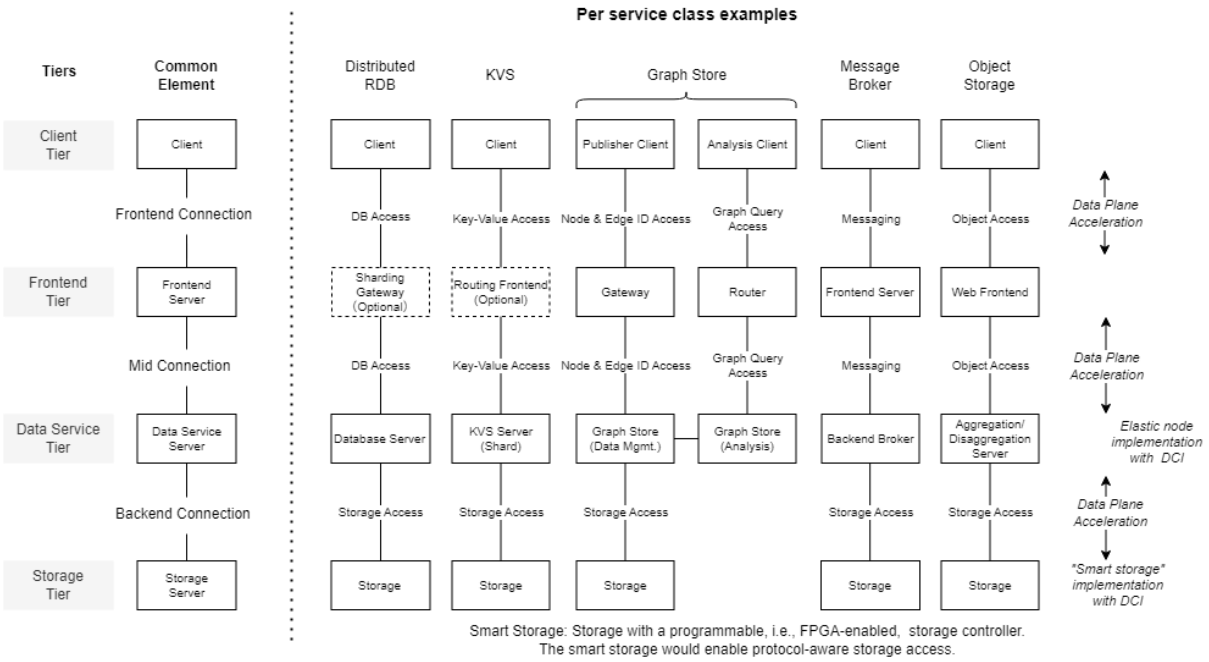

Figure 5-1-1, Generic Architecture Structure of the IDH Services

Each element included in the above architecture structure is explained below:

Components

- **Client:** Interacts with the IDH service to utilize required data securely.
- **Frontend Server:** Provides load balancing, authentication, authorization, service protocol translation, data access request routing, and data preprocessing such as format conversion, data aggregation, intelligent caching, logging, and some data services such as stream analysis.

- **Data Service Server:** Provides fine-grained data access controls, queries, full-range data processing such as long-range time series analysis and forecasting, model training of machine learning, data replication, data life cycle management, data-at-rest encryption, and logging.
- **Storage Server:** Provide data read and write functions, storage tiering, data protection e.g., mirroring and backup, data recovery, data de-duplication, and logging.

Connections

- **Frontend Connection:** connects a client and a frontend node.
- **Mid Connection:** connects a frontend node tier and a data service node.
- **Backend Connection:** connects a data service node and a storage node.

Two examples of connection patterns between the IDH components are shown in Figure 5-1-2. It should be noted that vertical and horizontal interconnections between frontend servers are supported when required, which is described as Example 2 in the figure. With this connection pattern, it is possible to well-support moving clients because the system can be configured so that the moving clients reconnect to another frontend server as they move, and the required session information is synchronized before, during, or after the re-connection. Also, this pattern can support the distributed deployment of functions across small far edges placed near the antenna and relatively large near edges that can be connected through multiple access networks. Please refer to Annex C for details on the expected architectural structure, or implementation model, of each IDH service.

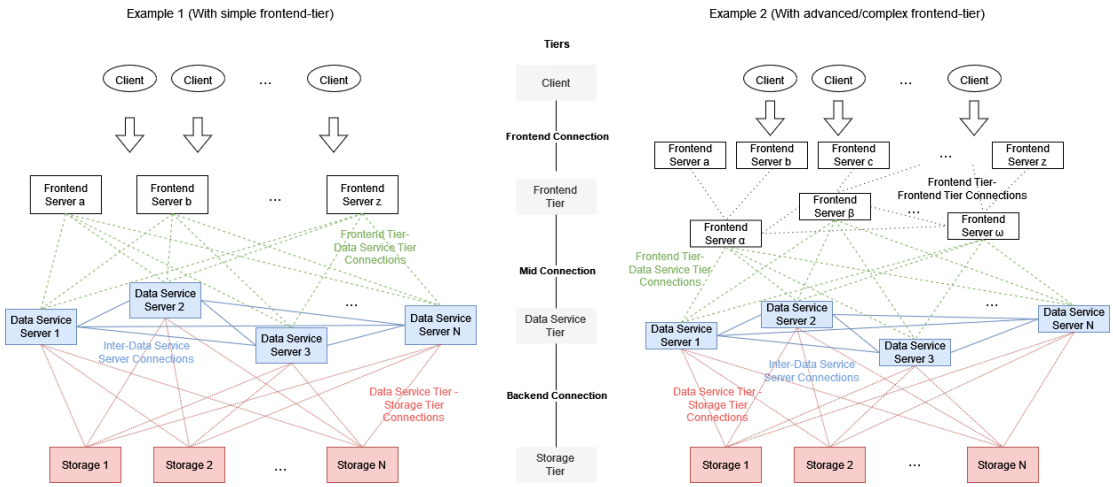

Figure 5-1-2, Connection Patterns between the IDH Components

5.2. Functional Architecture

Section 5.1 has shown the generic architecture structure of the IDH solution. In developing a concrete IDH solution, the next things to do are to identify the required functions and determine their appropriate placement within the IDH architecture structure, namely to define the IDH functional architecture, which is described in this section.

5.2.1. Data Plane Functions and Control Plane Functions

To identify the required functions, let's first consider two categories of functions, the data plane and the control plane.

The data plane functions are to store, process, and output data as requested by the user. As the IOWN Global Forum use case assumes high-efficiency and possibly real-time processing of large amounts of data, so as discussed in Section 5.1, the data plane's physical implementation will be a geographically distributed model.

On the other hand, the control plane functions are to manage the structure and state of the system, and control where and what data plane function should be executed and how data should be transferred between them. Such management and control functions can be implemented centrally, but even in that case, when building a system by on top of distant data centers, reference information of the control plane functions may be cached in each data center for real-time controls.

Figure 5-2-1-1 below visualizes an image of the relationship between such data plane functions and control plane functions.

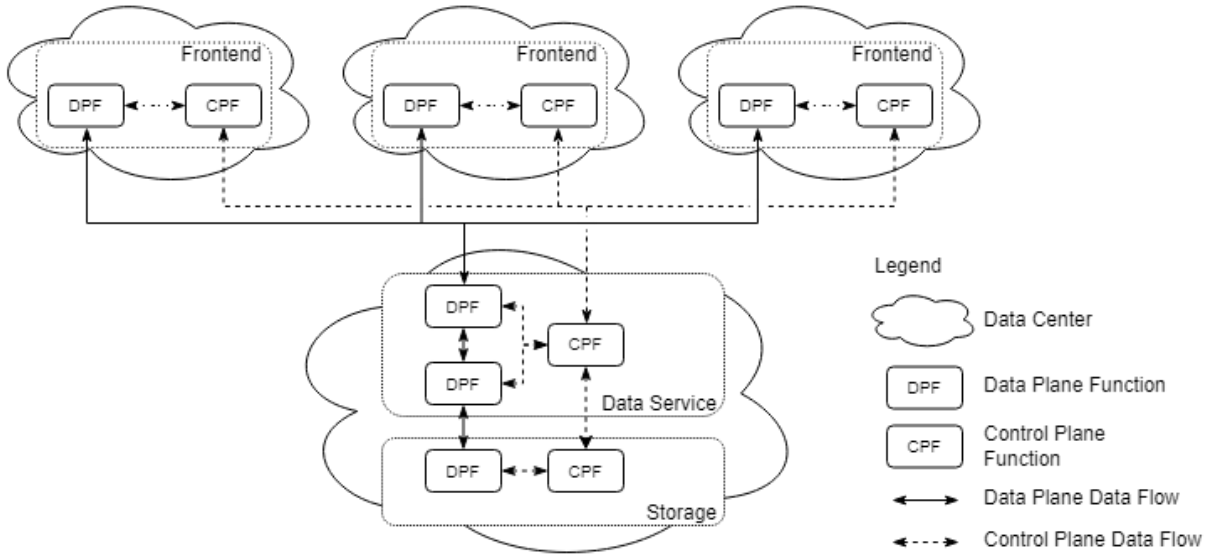

Figure 5-2-1-1, Data Plane and Control Plane Functions within an IDH Architecture Structure

5.2.2. Data Plane Functions and their placement in the IDH architecture structure

Detailed data plane functions to be implemented will depend on the IDH Services and their specification determined by each IDH Solution provider. However, it would be reasonable to consider the following set of common data plane functions to build effective, safe, and secure data management, processing, and sharing in a geographically distributed environment:

- Authentication and Authorization (AuthNZ)

  These functions authenticate and authorize client requests to an IDH service of concern. When authenticating the client requests, not only the validity of the credentials provided, but also the identifier of the client, the application context in which the request originated, and/or the presence of any required authorization from other clients would be checked. When authorizing the client requests, not only the called function, but also the scope of input data, data processing location(s), the details of called data processing function, and the content of output data would be checked. Authentication and authorization may be performed separately for each IDH component, or authentication and authorization information may be

exchanged between IDH components to ensure strict consistency of data access. In the IDH service that handles sensitive information, authentication and authorization information may have a validity period or be subject to revocation processing.

- Transaction

  This function is to update data adequately. For example, this performs the necessary exclusive control when there are simultaneous writes to the same data, or responds with the confirmed data at that timing to read access while accepting many writes in parallel.

- Continuous Query

  This function streams relevant data to registered client applications or embedded data processing for processing/analyzing data at short latency. When streaming the data to an application, some simple data preprocessing may be applied, such as filtering, threshold-based triggering. To do so, the client application needs to register themselves, a continuous query that describes data preprocessing, and timing/cycles of continuous query execution.

- Query

  This function provides relevant data to an requesting client application. To do so, the client application issues queries to the IDH service in order to receive only the necessary data. If the client only needs total values and/or statistical quantities, the query will include those arithmetic functions. To accelerate performance, the query function can be executed in a hierarchical manner or parallel on multiple servers.

- Query Orchestration

  This function is to determine optimal execution plans for queries that span multiple servers and control their execution. When making such a plan, this function considers data location based on indexes and statistical information, available resources on each server, bandwidth and latency of the network connections between servers, etc.

- Embedded Data Processing

  This function is to let the IDH service do the custom data processing defined by an IDH user inside, rather than letting IDH provide the data to a client application. Although this function consumes more computing resources within the IDH service, it greatly reduces the amount of data transfer and eliminates the need to transfer sensitive raw data outside of the IDH service.

- Federated/Wrapped Access

  This function is to streamline data accesses that span multiple IDH services. When an IDH client requests data that spans multiple IDH services, an IDH component that receives it understands the API specifications of other IDH services and forwards data access requests, such as queries, to other IDH services, which then send back the data.

- Data Transmission and Reception (Data TRX)

  This function is to send and receive data between the components of IDH: clients, front-end servers, data service servers, and storage servers. In order to send and receive data safely, authentication and encryption functions would be performed before this function.

- Data Caching

  This function is to hold the used data for a while in order to speed up continuous access to the same data. A policy, which would be specified by the IDH service provider or each user of the IDH service, determines what data is retained and for how long.

- Data Format Conversion

  This function provides data format conversions such as unit conversion, columnarization/row-wise conversion, en/decoding, and compression/decompression. This function allows application developers to focus on application logic development without worrying about a data format to be used, and infrastructure engineers to freely change or optimize it for compatibility and performance reasons.

- Data Replication

  This function is to performs multiple data replications based on rules. For example, this would write data to multiple storage devices synchronously to guarantee data persistence, or write data to multiple storage devices asynchronously across multiple data centers to speed up read access globally.

- Data Encryption/Decryption

  This function is to encrypt and decrypt both data-in-transit and data-at-rest.

- Read and Write Access from/to Storage

  This function is to read and write data from/to storage. When reading from or writing to a remote storage device, implementation of this function becomes two-layered: high-level and low-level.

There are several possible options in determining the placement of these functions within the IDH architecture structure, i.e., the functional architecture to be used for building an IDH service. This means that a function can be placed in the IDH frontend, in data service, and/or possibly in storage-tier. Also, it does not mean that all functions must be implemented, and the implemented functions will vary depend on the IDH service.

Figure 5-2-2-1 below shows four examples of such functional architecture options.

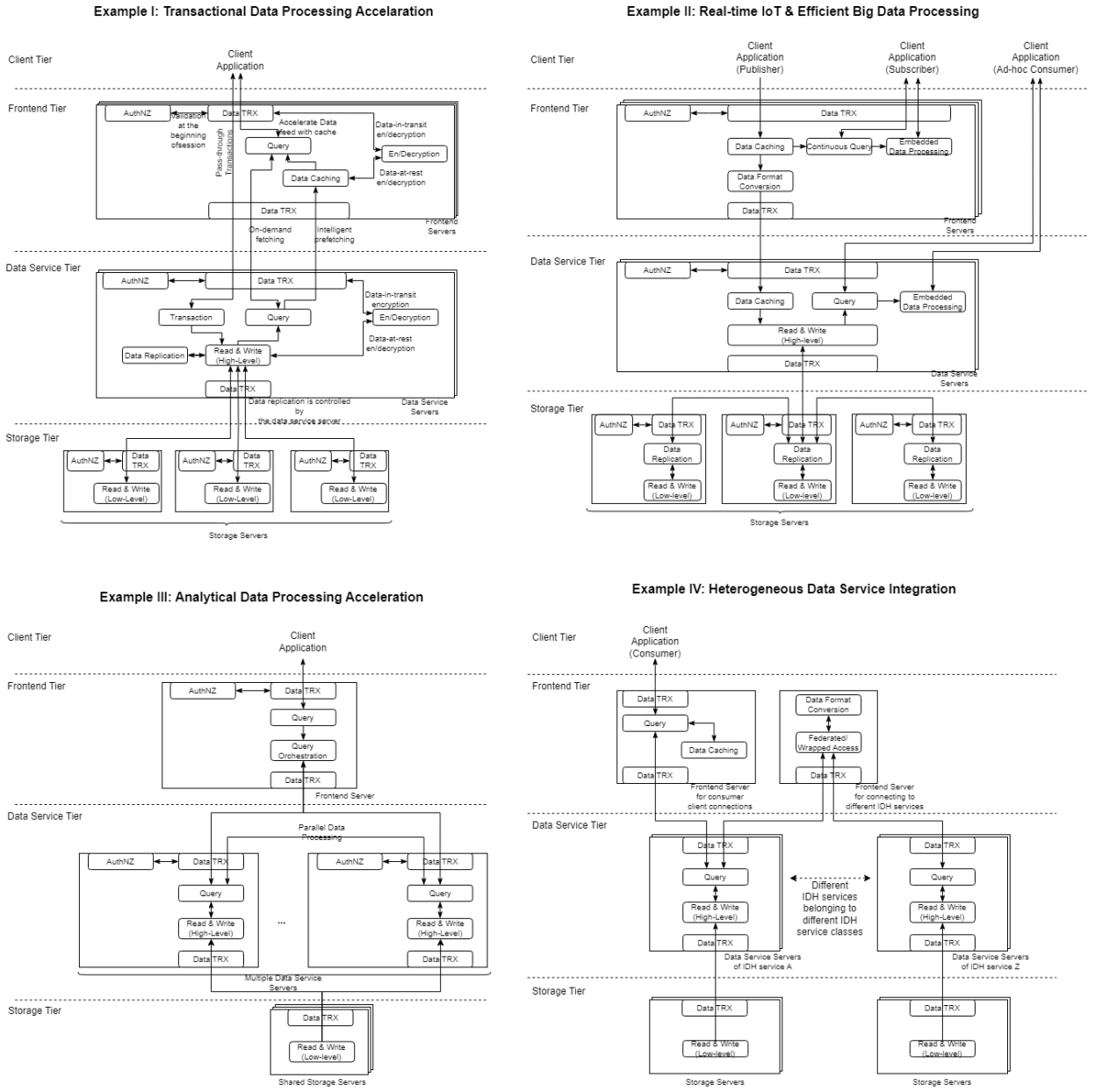

Figure 5-2-2-1, Functional Architecture Options at the Data Plane Layer

Functional architecture example I shown in Figure 5-2-2-1 is for an IDH service that provides high-performance transactional data processing. To accelerate read access performance, frequently-referenced data is intelligently cached in frontend servers located close to client applications. Also storage access parallelism is well controlled for accelerating reads and writes accesses by splitting data into multiple storage devices while guaranteeing data persistence.

Functional architecture example II is for real-time processing and efficient big data analysis of geo-distributed IoT data. To perform the required processing with low latency, intelligent frontend servers are placed close to the IoT device, in which required processing such as stream analysis is performed with minimal data movement. Also, to store IoT data

efficiently for a long period of time, collected data is periodically transferred to a large data center such as a cloud after being columnarized, and stored in an IDH storage layer that can efficiently persist data.

Functional architecture example III is meant to speed up data analysis by using multiple servers. For this reason, a frontend server that receives data analysis requests divides data analysis jobs appropriately and allocates them to multiple data service servers. Each data service server performs its assigned job, exchanging work-in-process data with other servers as needed, and returns results to the front end. Lastly the frontend server combines the response data from each server and returns the result to the client application.

Functional architecture example IV is for utilizing data across various IDH data services. For this purpose, a function is placed on a frontend server to handle federated or wrapping accesses to multiple other IDH services. Also, different IDH services will likely hold data in different forms. Therefore, in order to access the data efficiently, data format conversion will be necessary. Such conversion processing also takes place within the frontend server as needed. In this way, the burden of connection management and data conversion can be offloaded to multiple frontend servers especially

As these examples imply, the optimal function architecture or function placement will change depending on the IDH service. However, there seems to be several patterns that are used often especially for interconnections between IDH components. For example, efficient data transfer, reads and writes access to storage, query-to-query communication, etc. If more detailed specifications for these frequent patterns are clarified, and software components such as SDKs are well prepared and published, then, various companies, organizations, and communities will be able to start the development of interoperable IDH services and/or IDH service components smoothly. This will be the next job at the IOWN Global Forum.

5.2.3. Common Control Plane Functions and their arrangement in the IDH architecture structure

In addition to data plane functions, control plane functions are also required to realize IDH services, as any IDH service is a distributed system. Below are the major common control plane functions that are expected to be required to realize various IDH services:

- Identity Management

  This function is to manage identities required to access the IDH services, and also identities to be used to run the IDH services.

- Access Credential Management

  This function is to manage access credentials that are required for accessing the IDH services. Credential information is securely stored, refreshed periodically, issued only to authorized/verified IDH client applications, and revoked when necessary with this function.

- Metadata Management

  This function is to manage data about data, i.e., the structure, location, and status of disparate and distributed data in the IDH service(s), and feed them to a traffic director control plane function described below, as well as the query data plane function and the federated/wrapped access data plane function of the IDH service(s).

- Traffic Director

  This function is to determine the right destination and control traffic routing to the destination in a specific IDH service. Predefined rules, configuration information, and data catalog are referenced to route traffic to the right destination.

- Configuration Management

  This function is for managing the system configuration of the target IDH service(s). By knowing the configuration, the distributed execution of queries can be orchestrated, the status of each server can be

requested via multicast, and determine how to fail over in case of failure. In addition, configuration information can also be used to establish trusted connections between the IDH components, and verifying the IDH client applications.

- Configuration Change Management

  This function is to streamline changes in the system configuration of the target IDH service(s). For example, this function will instantiate the IDH service on a new server called a Logical Service Node in the IOWN Global Forum and update the routing information of the traffic director.

- Logging and Monitoring

  This function is to collect, analyze, and manage log data to monitor the IDH services as a whole. Continuous analysis of log data enables rapid detection or timely prediction of component-level failures, as well as a timely warning of system-wide anomalies, such as intrusions by attackers. Also, by keeping log data for a certain period of time so that when an incident occurs, the root cause and the scope of impact can be identified.

There are several possible options in determining the right functional architecture for these control plane functions, i.e., how these functions are placed in the IDH architecture structure.

For example, the control plane functions can be deployed at organization, regional/country, data center, resource group, IDH service, IDH service tier, and/or IDH component level. When the control plane function needs to be located in each IDH component, it would be implemented within a driver or as an additional agent installed on the server where the data plane runs. When the control plane function is located at other levels, they are instantiated at an independent server, or embedded in some IDH components like IDH data service servers.

Also, the interrelationship between these distributed functions can be a federation model in which all instantiated functions are treated as equal, or a hierarchical model such as master - submaster - slave relationship. Figure 5-2-3-1 shows some of such functional architecture styles for the IDH control plane functions.

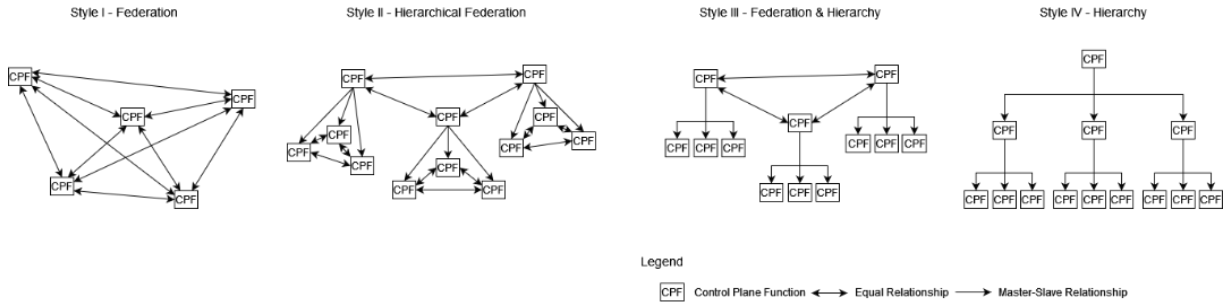

Figure 5-2-3-1, Possible Functional Architecture Styles for the Control Plane Functions

The level at which control plane functionality is deployed and the functional architecture style will vary depending on the control plane functionality and the IDH service implementer's decision. However, due to the characteristics of the IDH control plane, those options will be narrowed down. Table 5-2-3-1 below shows the expected deployment scenarios for each control plane function.

Table 5-2-3-1, Expected Designs of each Control Plane Function

Common Control Plane Function	Expected Highest Deployment (Instantiation) Level	Expected Lowest Deployment Level	Expected Lowest Deployment Level	Remarks

Identity Management	• Organization level	▪ Regional/ Country level, or • Data Center level	▪ Hierarchical, or ▪ Federation & Hierarchy	Identities are generally managed by each organization. Especially for large organizations, identities will be managed hierarchically according to the organizational structure. In inter-company/organization collaboration, identity management may be federated.
Access Credential Management	▪ Organization level, or ▪ Data Center level	▪ Regional/ Country level, or ▪ Data Center level	▪ Hierarchical, or • Federation & Hierarchy	Access credential management basically may have the same structure as Identity management because it issues credentials for accessing IDH services based on Identity management. However, once this function is hacked or abused, the system can be hijacked, which causes enormous damage to the organization. Therefore, this function may be implemented at a lower level such as data centers.

Metadata Management	▪ IDH service level	▪ IDH service level	▪ Federation, or ▪ Hierarchical	To implement metadata management in an IDH service, it would be possible to let each IDH component federate so that they can exchange the data about data they have with each other, or to put a specific metadata master server and let it manage all of the data about data. Some IDH services may exchange metadata with other trusted IDH services through federation. In this case, the additional federation layer will be built on top of the per-service metadata management layer. This streamlines data usage across IDH services.

Traffic Director	• IDH service level	▪ IDH service tier level, or • IDH component level	• Hierarchical	If the function for controlling destination and traffic routing between components within an IDH service is to be distributed and executed on each component, each component needs to understand the overall system configuration and operating status of the IDH service. To do so, federation can be used theoretically, but the multicast or broadcast traffic for state management can become overwhelming if there are a large number of components. Therefore, it would make sense to implement this hierarchically, in conjunction with the configuration management control functionality. Another way to implement this functionality is to have gateway servers within each tier of the IDH. The gateway server controls the destination and traffic routing of intra-tier and inter-tier communications. In either case, the style of functional architecture will be hierarchical.

Configuration Management	▪ Organization level, or • IDH service level	• IDH component level	• Hierarchical	A configuration master server needs to be instantiated at least per IDH service and manages the system configuration information of the target IDH service because otherwise IDH service cannot be operated properly. Each IDH component then reports its status to the master in a timely manner. Depending on the IDH service provider, it would be desirable to manage the configuration of all IDH services they operate in one place so that overall resource usage can be optimized. In that case, the master is instantiated at the organization level and manages the overall configuration of all IDH services deployed on distributed data centers.
Configuration Change Management	▪ Organization level, or • IDH service level	• IDH component level	• Hierarchical	Configuration change management basically will have the same structure as configuration management because configuration change management works on top of configuration management.

Logging and Monitoring	▪ Organization level, or • IDH service level	• IDH component level	• Hierarchical	Detailed logs should be collected and kept for unexpected incident. So each component has embedded control plane functionality for logging. Once data is collected, it will be analyzed to detect fraudulent activity, improve overall resource utilization, and so on. For this purpose, log analysis for the entire IDH service operating across multiple data centers is required. This means that log data will be analyzed hierarchically.

5.3. IDH Acceleration Mechanisms

There will be various bottleneck points in today's IDH-like services as described in chapter 4, for instance;

- When the client uploads or downloads a large amount of object data, such as a 10 GB log file, from some IDH service, client - frontend tier or frontend tier - data service tier connection becomes a bottleneck.
- For complex queries that need to shuffle or join a few PB data across multiple data service servers, inter-data service server connection becomes a bottleneck.
- For full scanning type queries that read a few EB data from the storage, data service Tier - Storage Tier connection becomes a bottleneck.
- Real-time data service is not easy especially when the client is located apart from the frontend servers. In today's cloud-based implementation models, basically, the frontend servers are located near the data service server to lower the burden of connections between the frontend servers and the data service servers. Therefore, the client must connect to the cloud that hosts the IDH services fast, otherwise, if the client is located far from the cloud data center. the real-time services are impacted (or delayed).
- Most of today's IDH services only provide connection-level security but do not provide fine-grained data access controls. Therefore, to protect sensitive and/or privacy information, a filtering logic must be implemented in the application. However, as such applications are developed and/or modified many times over time, a security bug may be created, and the leakage of important information tends to happen. In addition, once the data is exported, there is no mechanism to assure that data is used appropriately within an allowed scope. Therefore, most businesses and consumers share very summarized data only with other parties, which hinders open data usage and inevitably unable to deliver full social benefits.

To meet future digitization demands, these gaps must be eliminated. The common acceleration mechanisms that can be exploited in various IDH services to address these gaps are described in this section.

5.3.1. Acceleration Mechanism to eliminate the gap around Client - Frontend - Data Service Tier connections

When IDH services are used, the client has to connect to the data services, often through the frontend tier, and data access requests and response data are exchanged. To achieve near-real-time processing, the network latency for these communications must be shortened, and the bandwidth must be widened, especially when a large amount of data needs to be transferred repeatedly.

Such an objective is well-supported by the IOWN OpenAPN and the IOWN DCI architecture. The IOWN OpenAPN provides a direct optical communication path between two endpoints. Thus, the latency is shortened, and the bandwidth is widened. The IOWN DCI architecture controls a flexible and dynamic establishment of the high-speed network on top of the IOWN OpenAPN. It governs the inter-node communication within the cluster and the cluster-to-cluster communication through the IOWN DCI Gateway.

A control plane mechanism embedded in the IDH services is also essential to achieve the objective. As the data is distributed across multiple data service nodes, data access requests must be forwarded in consideration of data location. The IDH control plane will deliver the latest data placement knowledge to the clients and/or frontend tier. By knowing the data placement, the client and/or frontend can become smart* and directly send the data access requests to the most appropriate data service server. This reduces extra data transfer, thus can improve latency and bandwidth.

* After this, we will call the client that becomes smart a smart client.

5.3.2. Acceleration Mechanism to eliminate the gap around inter-Data Service Server communications

When data processing, such as complex join, simultaneous update of multiple records, etc., is triggered frequently, large amounts of data are exchanged among data service servers. If the network speed is slow, it wastes CPU resources on data service servers. That means, especially when the blocking protocol is used, as is often the case with today’s implementation models, CPU cycles will be wasted due to I/O wait. The non-blocking protocol could alleviate this issue, but it depends on a highly reliable network and has a problem of increasing complexity for distributed and asynchronous state management.

The IOWN OpenAPN helps eliminate such problems in both cases because it provides a high-speed and reliable network. On top of this, the IOWN DCI can establish such a high-performance IDH cluster dynamically combining multiple resources. In addition, IDH services will be designed to leverage the power of non-blocking protocols, i.e., RDMA in DCI, and to process and manage huge amounts of data in near real-time.

5.3.3. Acceleration Mechanism to eliminate the gap around Data Service Server - Storage Tier communications

In the IDH services such as Object Storage and Virtual Data Lake, the exchange of data between the storage tier and the data service tier is extensive. Relying on local storage to solve this problem is not desirable if considering Cloud-Native principles. That means modern system architectures should be determined to scale compute and storage separately on demand. In that sense, the storage becomes a type of remote server in the IDH reference implementation models. In many of today's cloud-based implementation models, the storage is built as such a server. However, these storages are not very smart. It just responds to the requester with the requested data blocks or objects. In today’s implementation models, the amount of data transferred at that time can be hundreds of terabytes with one request, and the system will be unresponsive for a long time. Therefore, today’s IDH services often consume a lot of the network bandwidth for communications between the data service servers and the storage.

The IOWN OpenAPN will solve this problem due to its high-speed network services established between data service servers and storage. In addition, the data transfer between data service servers and storage will be accelerated by

RDMA. This is because the TCP protocol that responds to Ack each time has a high overhead to transfer a large amount of data. Therefore, protocol level tuning is thoroughly done to increase the performance of the IDH services. Also, data preprocessing will be actively offloaded on the storage side using Smart NICs and the like in IDH services. By allowing data to be filtered and aggregated on the storage side, data transfer across the network will be reduced by a factor of 100 or more. With these mechanisms, IDH services will run much faster in the IOWN Global Forum reference implementation models than in today's implementation models.

5.3.4. Acceleration Mechanism to eliminate the gap that hinders the realization of real-time services

With today’s cloud-based implementation model, the client always connects to the data service located in the central cloud, and data access requests might travel over a long distance, which increases the latency, lowers the bandwidth, and worsens the energy efficiency.

With the IDH implementation model, it is assumed that the frontend tier would be placed near the client, possibly embedded in the client application, or placed in the regional edge clouds in a distributed manner so that the above issues are mitigated.

To well support such an implementation model, the IDH frontend tier is equipped with the required data plane functionality, such as temporal data pipelining, intelligent caching, primary data processing, data protection, and asynchronous data propagation to the data service tier in other regional edge clouds and/or the central cloud.

5.3.5. Acceleration Mechanism to eliminate the gap around data security and privacy protection

In the fully digitalized era, various data will be used and exchanged so that business productivity, personal experience value, and social benefit are increased. However, security and privacy concerns may slow down such a transition. To accelerate the transition, a more secure and trustworthy data management and sharing infrastructure must be built.

The IDH solution framework will provide a mechanism for robust security and privacy, applicable across various implementation models such as the Distributed RDB, the KVS, the Object Storage, and the Message Broker. With this security mechanism, all data accesses are authenticated with unified or federated identity management, authorized based on common policy management, and response or output data are guaranteed so that it complies with fine-grained access controls.

It should be noted that only high-level security features are explained in each IDH service section of Annex C in this document. Detailed security features and architecture will be explained in a separate upcoming technical paper.

6. Business Value of the IDH Solution

In order to realize future digital experience services, it will be necessary to process a wide variety of large amounts of data, sometimes with low latency as described in Chapter 2. To meet such requirements, various data hub services with features as described in Section 3 will need to be developed. However, due to the gaps described in Chapter 4, it would be very difficult to achieve the required low latency. In addition, if the system is built ignoring the latency requirement, its cost and energy consumption will be too large, which would not be acceptable, in today's technology landscape.

That's why we need a new solution: the IOWN Data Hub. Its functional architecture described in Chapter 5 supports operation in a geo-distributed environment, and its acceleration mechanism dramatically improves processing performance. Therefore, with IDH, various digital experience services that require very high real-time-ness can be realized at low cost and low energy consumption.

Please refer to Annex D for details on how various digital experience services can be realized with the IDH solution.

7. Next Steps

In this document, the data management requirements for the future use cases envisioned by IOWN Global Forum and the gaps in today's implementation technology are analyzed first, and then how to build the IDH services to solve the gaps and meet the requirements are discussed.

However, there is still much work to be done in order to actually use IDH to solve various problems. For example, it may be necessary to first conduct a set of primitive PoCs, which is described in the IDH PoC Reference document [IOWN GF IDH PoC], and then perform advanced PoCs that are more specialized for assumed use cases. Also, as data security and data sovereignty become more critical requirements today, it is necessary to develop an appropriate implementation model of IDH services to meet these requirements. And finally, it will be necessary to build an open ecosystem to accelerate innovation around the IDH. To that end, it will be necessary to further refine the functional architecture described in Chapter 5 so that various companies and organizations can develop each IDH component in parallel and integrate them quickly without having overhead.

IOWN Global Forum plans to continue these efforts and publish related technical papers once these are conducted.

References

[IOWN GF AIC UC]	IOWN Global Forum, “AI-Integrated Communications Use Case Interim Report,” 2021. https://iowngf.org/use-cases/
[IOWN GF CPS UC]	IOWN Global Forum, “Cyber-Physical System Use Case Interim Report, Version 2.0,” 2021 https://iowngf.org/use-cases/
[IOWN GF DCI]	IOWN Global Forum, “Data-Centric Infrastructure Functional Architecture,” 2022.
[IOWN GF Open APN]	IOWN Global Forum, “Open All Photonic Network Functional Architecture,” 2022.
[IOWN GF RIM]	IOWN Global Forum, “Cyber-Physical System, Reference Implementation Model,” 2022
[IOWN GF IDH PoC]	IOWN Global Forum, “IOWN Data Hub PoC Reference”, 2022

Appendix A: Service Class Details

A.1. Basic Service Class Details

A.1.1. Distributed Relational Database

A Distributed Relational Database (Distributed RDB) is a basic service class for storing structured data. A Distributed RDB is ideal for use cases where data is “well-formed,” (meaning the individual data can be identifiable by some kind of structural query) and the data read/write speed is sufficient. This includes applications that can store and use data without any extra data conversion process outside the Data Hub. This service class is intended for online transaction processing (OLTP) in general databases and data analysis use cases such as online analytical processing (OLAP) in data warehouses.

Today’s typical Cloud-based Distributed RDB can handle 20 ~ 50 K transactions per second (TPS) for OLTP and much less for OLAP. Some distributed databases can provide better performance by having more servers and distributing the data based on an algorithm. However, cross-server data queries, especially joins, do not scale linearly.

If the performance requirements of the target application fall within this range, then it can be covered by this service class. On the other hand, if the use case requirements cannot be met in terms of performance, this service class cannot be used, and it is necessary to use other service classes such as Key-Value Store or Message Broker to receive the data today.

Potential usage scenarios

- Machinery status and manufacturing order data management for the CPS: Industry Mgmt. - Process Plant Automation use case
- Network device status data management for the CPS: Network Infrastructure Mgmt. - Network Device Failure Prediction use case

Functional requirements

- Online transaction processing (OLTP)
- Online analytical processing (OLAP)
- Sharding for scalability with configurable consistency levels
- APIs

  The IDH Distributed RDB will be equipped with at least one, and ideally all, of the following APIs:

  - SQL - as a data definition and data manipulation language standard
  - SDK/Driver-based APIs, such as JDBC, ODBC, etc., that can publish SQL
  - REST APIs that accept SQL-based data queries and respond in JSON format

Non-functional requirements

- Scalability of data writing and reading speed when the amount of data to be stored increases.
  - Especially sufficient speed for writing data when the data generation interval is very short.
- Strong consistency even in a Distributed RDB.
- Flexibility to change consistency level according to performance requirements.

A.1.2. Key-Value Store

A Key-Value Store (KVS) is a basic service class for storing an enormous number of data records, e.g., trillions, quadrillions, or more, although the size of each record is often quite small (e.g. less than 100 KB). For example, time-series data and sensor data would be handled by this service class. It should be noted that the value field of a KVS could be JSON or any other wide-column type field, that consists of various sub-key (child, grandchild, or descendant key) and value sets.

Potential usage scenarios

- Sensor data management for the CPS: Area Mgmt. - Disaster Notification use case
- Telemetry data management for the CPS: Network Infrastructure Mgmt. - Network Device Failure Prediction use case
- Health record management for the CPS: Healthcare Mgmt. - Disease Outbreak Prediction use case
- Power balance status data for the CPS: Smart Grid Mgmt. - Renewable Energy Flow Optimization use case
- Audience position and posture data management for the AIC: Entertainment - Interactive Live Music use case

Functional requirements

- Key-based data access/manipulation
- Sub-key-based data query
- Sharding for scalability with configurable consistency levels
- APIs

  The KVS will be equipped with at least one, and ideally all, of the following APIs:
  - REST APIs to
    - put and update data, specifying the key and the value, or a list of key-value sets
    - get and delete data, specifying a single key, or a list of keys
    - query data, specifying query conditions against key and value data
  - REST-like APIs to
    - do the same things as above REST APIs do
    - do the transaction (an example is MongoDB Data API)
  - SDK/Driver-based APIs to:
    - do the same things as above REST APIs do
    - do the transaction (an example is MongoDB Drivers API)
    - support various languages such as Java, Python, C/C++, Node.js, PL/SQL, etc.

Non-functional requirements

- Consideration of data location during data access
  - Note: Data will be distributed across multiple KVS servers. So, any data access requests need to be forwarded to the node where the target data exists. If there are multiple copies of data, then the nearest node should be selected for performance.
- Latency (between data storage and retrieval)
- Depending on the performance and size requirements, we should be able to choose data storage from different storage classes, such as
  - In-memory KVS without persistence,

- In-memory KVS with persistence, and
- Disk-based KVS.

A.1.3. Graph Store

This basic service class stores data modeled as a graph. Nodes and edges that connect nodes form a graph. Information of nodes is stored as properties of those elements. The stored data forms a graph of information with links between pieces of information.

There are two major variations of the Graph Store, one is called Resource Description Framework (RDF) store, and the other is called Property Graph Store. In the RDF store, stored data might follow a pre-defined schema named ontology. The ontology describes how the data is structured and allows the application developers to formulate a semantic query. A semantic query allows for retrieving information based on the associative and contextual scope. It shall be noted that there is a mechanism, called web ontology language (OWL), to support interoperability between ontologies developed by different parties. In the Property Graph Store, graph data is managed in a simple manner so that data can be traversed easily. However, there is no mechanism to support interoperability.

Potential usage scenarios

- Road structure data management for the CPS: Mobility Mgmt. - Energy Optimal Routing use case
- Power grid structure data management for the CPS: Smart Grid Mgmt. - Renewable Energy Flow Optimization use case

Functional requirements

- Graph data manipulation (create, update, read and delete)
- Graph data query that analyzes the linkage between the nodes and the edges
- Data model and data manipulation functions to support RDF, RDF schema, and OWL standards
- Data model and data manipulation functions to support Property Graph standards
- APIs

  The IDH Graph Store will be equipped with at least one, and ideally all, of the following APIs:

  - o REST API that accepts queries based on SPARQL for the RDF store, and at least one of PGQL, G-CORE, Cypher, or GQL for the Property Graph Store, and responds in JSON format
  - o SDK/Driver-based APIs, such as JDBC and ODBC that support SPARQL, PGQL, G-CORE, Cypher and/or GQL for various programming languages
- Other key requirements
  - Ontologies and metamodel need to accommodate the semantics of a specific use case and across use cases, such as metamodel, cross-domain ontology, and domain-specific ontologies.
  - A domain-specific language to query various information, such as geographical information, and/or history information
  - Synchronous data retrieval, i.e., query-response interactions
  - Asynchronous data retrieval, i.e., subscribe-notification interactions
  - APIs need to be enhanced so that create-read-update-delete (CRUD) operations can be supported. Such a concept is currently under discussion in the ISO/IEC JTC 1/SC 32/WG 3 and will be standardized as Graph Query Language (GQL) in the upcoming years.

Non-functional requirements

- Scalability of the system to accommodate large graph of data
- Mechanisms to avoid retrieval of overly large subgraph resulting in query timeout, e.g., pagination

A.1.4. Message Broker

A Message Broker is a basic service class intended to be used as a message brokering service or data streaming service for any of the following scenes:

1. The application needs to be distributed to a large number of clients.
2. Data contains a mixture of sizes, types, or formats.
3. Data needs to be formatted, converted, or aggregated in a complicated pipeline.

Potential usage scenarios

- Event data collection from various machinery, equipment, and robots in an ad hoc fashion for the CPS: Industry Mgmt. - Process Plant Automation use case
- Power supply, demand, and flow data collection from various smart meters, switchgears, and transformers for the CPS: Smart Grid Mgmt. - Renewable Energy Flow Optimization use case
- Data pipelining to collect various data in an ad hoc fashion from various sources for the CPS: Society Mgmt. - Sustainable Society use case
- Message pipelining for the AIC: Human Augmentation - Mind-to-Mind Communications use case

Functional requirements

- Pub/Sub-style message delivery
- Push-based message delivery if matched with certain conditions
- Support of various message formats/protocols, such as MQTT, JMS, JSON document, etc.
- APIs

  The Message Broker will be equipped with at least one, and ideally all, of the following APIs:

  - REST APIs to publish and receive messages to/from the Message Broker
  - SDK/Driver-based APIs (e.g., Apache Kafka/Pulsar compatible API) to publish/receive/subscribe messages to/from the Message Broker
  - Lightweight protocols running over TCP or UDP, such as gRPC, MQTT, AMQP, STOPM/TTMP, AMUDP, etc. to publish/subscribe messages to/from the Message Broker

Non-functional requirements

- Auto-scaling when flow rate changes
- High throughput (amount of data that can be relayed)
- Low data transfer latency
- Maintaining a good performance regardless of data size, type, format, or the number of publishers/consumers
- Ensuring data reachability

A.1.5. Distributed File Storage

Distributed File Storage is a basic service class to store the file in a hierarchical and scalable manner at a medium cost. Examples of such a Distributed File Storage system existing today are HDFS, BeeGFS, Lustre, GlusterFS, and, of course, file storage services of various clouds. One typical use case of such a Distributed File Storage solution is to store documents that have to be shared among people within an organization. Another use case is to share data and log files that must be shared across various servers and/or systems.

Potential usage scenarios

- Log data management for the CPS: Network Infrastructure Mgmt. - Network Device Failure Prediction use case
- Learning content data development for the AIC: Remote Operation - Professional Learning use case
- Interaction data management for the AIC: Human Augmentation - Another Me use case

Functional requirements:

- File storage mounting
- Auto versioning
- Locking management
- De-duplication of files (i.e., Single instance storage), which automatically eliminates duplicated/equivalent files and keeps only pointers for deleted files
- APIs

  The Distributed File Storage will be equipped with at least one, and ideally all, of the following APIs:

  - Well-established file access APIs, such as NFSv4, CIFS/SMB, etc.
  - APIs to support file sharing over the HTTP/Internet, such as WebDAV, etc.
  - APIs to support lightweight file sharing over TCP or UDP, such as sFTP, UDP-based Data Transfer Protocol (UDT), etc.
  - Hadoop Compatible File System API

Non-functional requirements

- Maintaining performance when multiple mounts are made at the same time
- Scalability (amount of data that can be stored)
- Ensuring data reachability

A.1.6. Object Storage

Object storage is a basic service class to store relatively large object data, such as video, image, log, and other types of semi-structured and unstructured data, at a low cost. Typical use cases are to store big data for a more extended period of time and provide data-to-data analysis applications such as AI and data lakes to prepare for future data utilization.

Potential usage scenarios

- Big data management for training inference models for anomaly detection for the CPS: Industry Mgmt. - Factory Remote Operation use case
- Data exchange platform, to collect various detailed data required for the smart grid proactive controls, such as weather forecast, factory operation plan, etc., for the CPS: Smart Grid Mgmt. - Renewable Energy Flow Optimization use case
- Learning content data management for the AIC: Remote Operation - Professional Learning use case

Functional requirements:

- Object operations to create, update, read, and delete an object
- Multi-part object upload and download
- In-storage query to extract the filtered data only (optional)
- Inter-region object replication

- Lifecycle management to automatically move old data from hot storage to cold/archive storage, or delete them
- Gateway function to expose the Object Storage as a file system
- Storage-side encryption and encryption key management
- APIs

  The Object Storage will be equipped with at least one, and ideally all, of the following APIs:

  - General Object Storage APIs (e.g., REST API that is compatible with AWS S3)
  - Expanded APIs for various applications: classic file access like CIFS/SMB/NFS, high throughput and efficient data upload/download, specifying query conditions, etc.

Non-functional requirements

- Throughput for handling the massive volume of data ingestion and retrieval
- Dynamic scalability for accommodating fluctuating data flow
- Flexible to well balance performance and cost

A.2. Applied Service Class Details

A.2.1. Context Broker

This service class supports context-aware communication between data providers and applications. Data is requested by specifying the context scope, where context identifies the focus of the information, such as a thing (e.g., a specific building), a type of thing, geographic scope, temporal scope, or a combination of these. The typical architecture of the Context Broker is a federation of many context providers that have access to data, or federation of Context Brokers, or a hybrid combination of providers and brokers. Upon a context request, the Context Broker provides the context managed locally and the context managed by other Context Brokers and context providers. Thus, this system acts as a broker that provisions data to the requester. The data provisioning might happen in two ways: 1) a synchronous request is responded to with a single data response, and 2) an asynchronous request elicits the establishment of a requester for a stream of information that matches the requested context. As part of the context, a piece of information might be related to other information forming an interrelated context.

A Context Broker inherits the following basic service classes, with expanded APIs and features to support federation and/or aggregation of services:

- Message broker
- Distributed Relational Database or KVS
- Graph Store

Potential usage scenarios

- Various data collection and contextualization for the CPS: Healthcare Mgmt. - Disease Outbreak Prediction use case
- Various data collection and contextualization for the CPS: Society Mgmt. - Sustainable Society use case

Functional requirements

- Same requirements as inherited service classes
- Continuous and secure data collection and linkage

Non-Functional requirements

- Same requirements as inherited service classes

A.2.2. Converged DB

To satisfy multiple IOWN Global Forum use cases, different types of data need to be linked and analyzed simultaneously, as described in various use case scenarios in Annex D. For instance, when considering a scenario in which people's behavior inside a building is analyzed to provide appropriate digital support for them, the interactions of people (including person-to-person or person-to-thing) must be managed and processed together. Therefore, the data structure that expresses such interactions requires great flexibility. If different IDH services are used for different data structures, the required real-time results may not be achieved, especially when considering the process of reading them in sequence. A Converged DB provides the foundation for high-speed simultaneous processing of different types of data in one place to meet such challenging requirements.

In that sense, a Converged DB inherits the following basic service classes:

- Distributed Relational Database
- Key-Value Store
- Graph Store
- Message Broker

Potential usage scenarios

- Situation data management for the CPS: Area Mgmt. - Security use case
- Traffic situation data management for the CPS: Mobility Mgmt. -Safety Maneuver

Functional requirements

- Same requirements as inherited service classes
- Extended functions to manipulate different kinds of data together

Non-Functional requirements

- Same requirements as inherited service classes
- QoS/Resource allocation controls depend upon the usage

A.2.3. Virtual Data Lake (Federated Storage)

The concept of a data lake, which was born with the advent of Hadoop, is now being transformed so that cloud-based storage such as Object Storage can be effectively supported. As the amount of data increases continuously, the data lake solution shall change to a distributed model based on multi-cloud, edge cloud, etc. To prepare for such expected future demands, the IOWN Global Forum envisioned a new concept, the Virtual Data Lake, which provides a mechanism for bundling geo-distributed storages and making them appear as one.

In that context, the Virtual Data Lake will inherit the Object Storage service class and/or the Distributed File Storage service class, with expanded APIs and features to support federation and/or aggregation of services:

Potential usage scenarios

- Various open data integration for the CPS: Area Mgmt. - Disaster Notification use case
- Data sharing platform, to share various detailed data required for the smart grid’s proactive controls, such as weather forecast, factory operation plan, etc. for the CPS: Smart Grid Mgmt. - Renewable Energy Flow Optimization use case
- Various log data integration for the CPS: Society Mgmt. - Sustainable Society use case

Functional requirements

- Same requirements as inherited service classes

- Federated secure access to the Object Storage and/or the Distributed File Storage, which might be geo-distributed and owned by different parties.

Non-Functional requirements

- Same requirements as inherited service classes
- Integration with federated identity management services

A.2.4. Virtual Data Lake House

The IOWN Global Forum anticipates a future in which multiple companies and organizations exchange big data openly and securely to solve social issues. For example, to stabilize the operation of the electric power grid in which renewable energy accounts for the majority of the supply, the supply and demand architecture of electric power will need to be exchanged in real-time and has defined an extended concept called the Virtual Data Lake House.

The Virtual Data Lake House is designed to provide secure and transparent access to various data stored in the geo-distributed heterogeneous data sources belonging to different owners with the support of the Data Catalog. It also orchestrates data processing to execute processing as close to the data source as possible to optimize performance. It does this by considering the placement of data and the IDH services.

The Virtual Data Lake House inherits the following service classes, with expanded APIs and features to support federation and/or aggregation of services:

- Distributed Relational Database
- Key-Value Store
- Graph Store
- Message Broker
- Distributed File Storage
- Object Storage

Potential usage scenarios

- Integration of various manufacturing data, such as machinery parameters, machinery operation, sensor logs, captured images, etc., for the CPS: Industry Mgmt. - Process Plant Automation use case
- Integration of various healthcare-related data, such as the number of patients visiting hospitals, symptom statistics, weather conditions, etc., for the CPS: Healthcare Mgmt. - Disease Outbreak Prediction use case

Functional requirements

- Same requirements as inherited service classes
- Extended functions to manipulate various data together that is stored in different IDH services
- Federated secure access to the geo-distributed and multi-party IDH services

Non-Functional requirements

- Same requirements as inherited service classes
- Integration with federated identity management services

Appendix B: Today’s Implementation Models of IDH-like Services and their Gaps

In this section, typical implementation models for today’s IDH-like services, and identified gaps against the IOWN GF use cases are explained. It should be noted that this analysis is only conducted against basic service classes, as most of today's services are categorized as basic service classes. Although some today’s services can be categorized as applied service class, they are outside the scope of this document because they are implemented with accepting limitations in scalability and/or application of accelerators.

B.1. Distributed Relational Database (Distributed RDB)

Existing Technologies Overview

Today’s Distributed RDBs are configured to have all-active masters on top of the shared storage. In such an implementation model, any master node can update and reference any data. Another example is to use a Read-only Replica to increase read-throughput. In this configuration, the replica node cannot update data but can provide read-only access. Besides these examples, there are also implementation models based on shared-nothing architecture. In these models, data is divided and assigned to each node or always synchronized among the nodes.

Today’s Implementation Models

Today’s typical implementation models are described in the figure B-1-1. Data access requests and responses to data access requests are forwarded in consideration of the distributed environment, as shown in the figure. Data access security is generally controlled by permissions established when a DB connection is made. This controls which tables and views can be accessed through that connection.

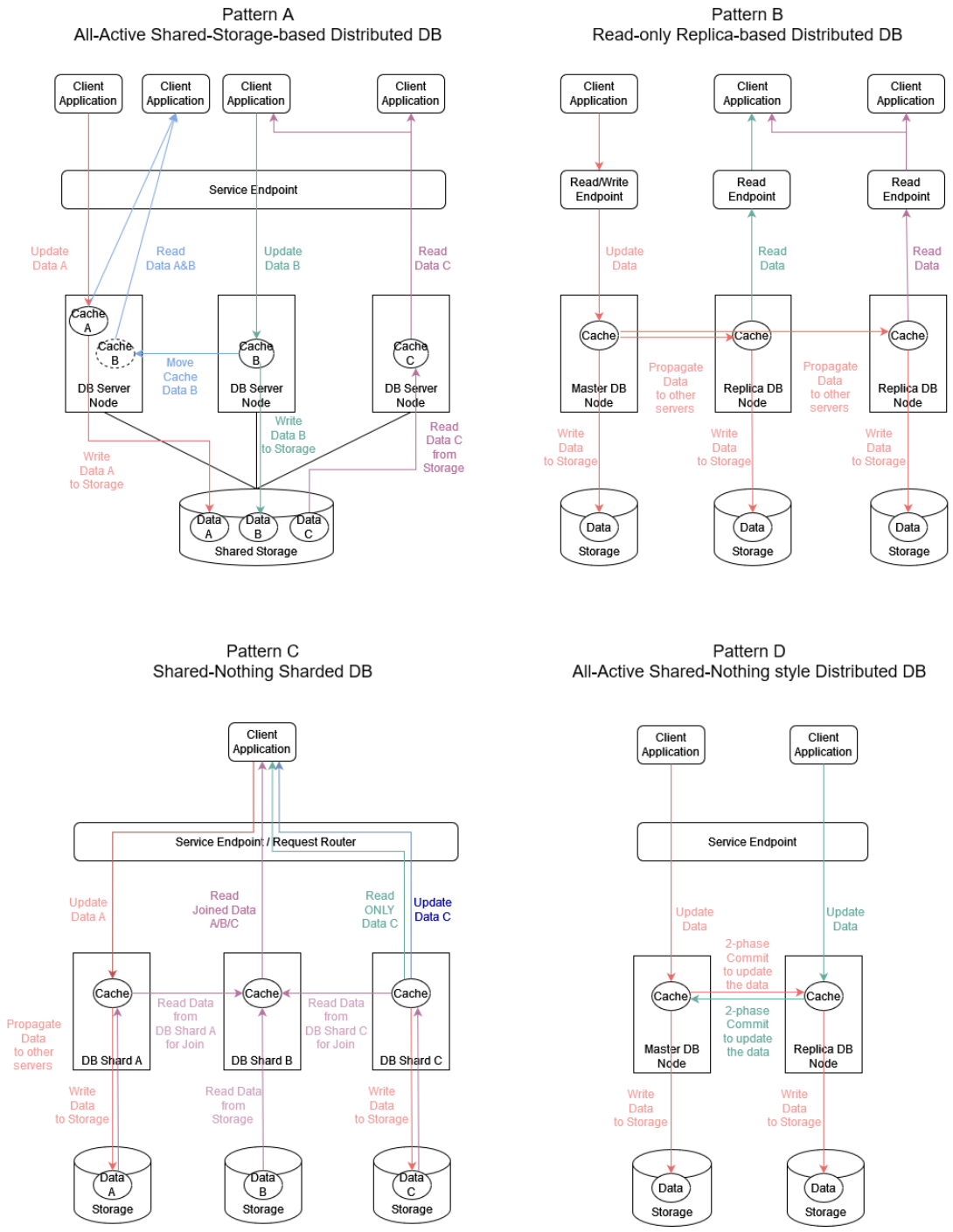

Figure B-1-1, Today's Implementation Models of the Distributed RDB

Gap Analysis for the DB Server Tier

A common consideration point across different implementation models of the Distributed RDB is the usage of a gateway that routes client requests to an appropriate instance of the Distributed RDB service in the cloud. In a cloud service, a special resource pool or server cluster tends to be used to run the distributed DB services for performance, and software-defined controls are embedded there to provision the DB instance quickly for the tenant. Therefore, the roles of the gateway are to provide the endpoint that can be used to access the DB, block any unauthorized access to the

DB, and provide load balancing and failover controls. Such a logical mechanism on top of the physical resource pool slows down the response time of the DB, which could be more than a few milliseconds even for the single record query in typical cloud offerings.

For implementation models like the All-Active Shared-Storage-based Distributed RDB, the performance will be degraded if a lot of cached data needs to be moved repeatedly. To avoid such a situation, the number of DB servers for a single application is typically 4 or 8. Thus it is usually very difficult to support as many as 1 million update transactions every second.

When assuming the digital era as described in the IOWN Global Forum use cases, the size of each data record will increase, and multiple clients will continually request access to data records. The result will be more frequent cached data movement resulting in network traffic that will continue to grow larger and larger.

To address this challenge, there will be several options for developing new technologies. This might include new network technology that reduces network latency when connecting DB servers and storage nodes and increases the amount of bandwidth to move cached data more smoothly. Another option might be to avoid moving cached data across servers and forward the data access request, namely the SQL statement, to the server where the requested data exists. This is especially ideal when the data access is larger than the request itself or many “chatty” update transactions. By doing so, it will be possible to utilize more DB servers to scale out the performance and make DB servers span across adjacent data centers, i.e., availability zone in the public cloud case, for durability.

The master nodes that accept updates quickly become a bottleneck for implementation models like read-only replica-based Distributed RDB. This happens because the RDB usually is expected to guarantee data consistency, so only the master can update the data. If the data is updated in different servers in an optimistic way, it becomes difficult to ensure consistency since data update conflicts and/or wrong data updates by referencing outdated data will occur quite frequently. In addition, it is also difficult to guarantee read consistency, e.g., the query requesting the latest data will be responded to older data by the RDB server, or dirty reads may happen as the data may be updated during the read transaction uncontrollably. To avoid such a situation, it is required to let the client ask the master node about committed transactions before it reads the data from one of the replicas and maintains the multi-version of data with a time stamp based on the most lagging local clock in the server cluster. Furthermore, if the replica server is placed in the adjacent data center, as in the public cloud today, data synchronization becomes a further burden to performance. Therefore, the read performance scales very well with this implementation model to one hundred of thousands of TPS. Still, the write performance will be limited to tens of thousands of TPS, thus not enough for the IOWN Global Forum use cases.

To increase the write-operation throughput, a technique known as a sharded DB is also used. In the sharded DB, the whole data is split into several units and distributed across multiple DB servers (Shards). When updating the data, the DB shard that manages the target data is responsible for updating operations. When reading the data, multiple DB Shards work together to process the query and respond to the data back to the client. In this way, the write throughput can be linearly increased by adding more servers if the data update request is evenly distributed among the Shards. However, this pattern is not perfect. For instance, the data exchange between Shards will be frequent if the client requests a complex query that includes complex joins, sub-queries, etc. As a result, the query performance will be extremely slow in today’s cloud environment. Also, if a server failure happens, specific data will not be accessible until the proper failover operation is conducted.

An All-Active technique can further increase the write performance and improve service continuity by eliminating any single point of failure. In this configuration pattern, data is always synchronized between 2 or several servers, and each server can conduct write operations with the support of the two-phase commit protocol. In this way, multiple CPU cores from multiple servers can work together to update the same set of data to increase write throughput. However, communications between/among servers will be very frequent in a “chatty” manner. The network will limit communication performance, and CPU usage efficiency will decrease due to the waiting time for communications.

It should be noted that these four patterns can be combined to solve a specific problem. For instance, Oracle RAC Sharding inherits patterns A and C, GCP Spanner inherits Pattern B and C, and Oracle TimesTen Scaleout inherits

patterns B, C, and D. In any case, the performance of these RDB implementations is significantly suppressed by the latency and bandwidth of network communications between DB servers, and between DB servers and data stores.

In addition, with today's cloud-based implementation models, clients have to connect to the cloud data first. As the number of cloud data centers is still limited, the latency to connect to the cloud may not be small which may exceed 5 milliseconds for remote clients. This hinders the realization of real-time services.

Lastly, it is required to consider security. Basically, almost all Distributed RDBs are equipped with some basic security features which can be used to manage access rights of DB users who have opened a DB connection to DB objects, such as table, view, stored procedure, etc. However, these are not enough if considering applications that handle very sensitive information.

For example, granular data access controls considering actual users are not possible in almost all Distributed RDBs, because DB connections between the application server and the DB server are generally pooled for performance, and they are shared by actual users who log on to the application server. In other words, such granular data access control that considers the actual user is performed on the application server, and thus risk of information leakages due to bugs and hacks remains large today.

Another example is a data-centric row and column-level access control. In this case, it might be required to restrict access to columns containing sensitive information if the relevant row is marked confidential. In such a situation, it is adequate to give data access rights only to specific users who are granted review privileges at the time. Currently, almost all Distributed RDBs do not implement such a detailed access control that considers both the content of data and the role of the user who actually accesses the data.

The last example is a case in which access rights to confidential information can be granted for statistical analysis purposes, but output/export of the confidential information itself from the RDB shall be prohibited. It can be said that there is no Distributed RDB that can control and manage such data usage rights.

Gap Analysis for the Storage Tier

Block storage is typically used as the data persistence layer for database-type workloads in today's cloud environment. To guarantee data persistence, the data is replicated locally three times across multiple storage servers. The storage servers handle the storage I/O requests sent from any clients and manage the volume and data blocks inside them while highly isolating each tenant's data for security reasons. To manage the system configuration and the address of each data block, the storage server embeds the internal DB, and updates the information when the configuration is changed. To increase the throughput and reduce latency, the Block storage service is located in each Cloud availability zone, and sometimes the multi-tenant control plan functionality is offloaded to the Smart NIC. A typical implementation model of today's Block Storage system is shown in figure B-1-2.

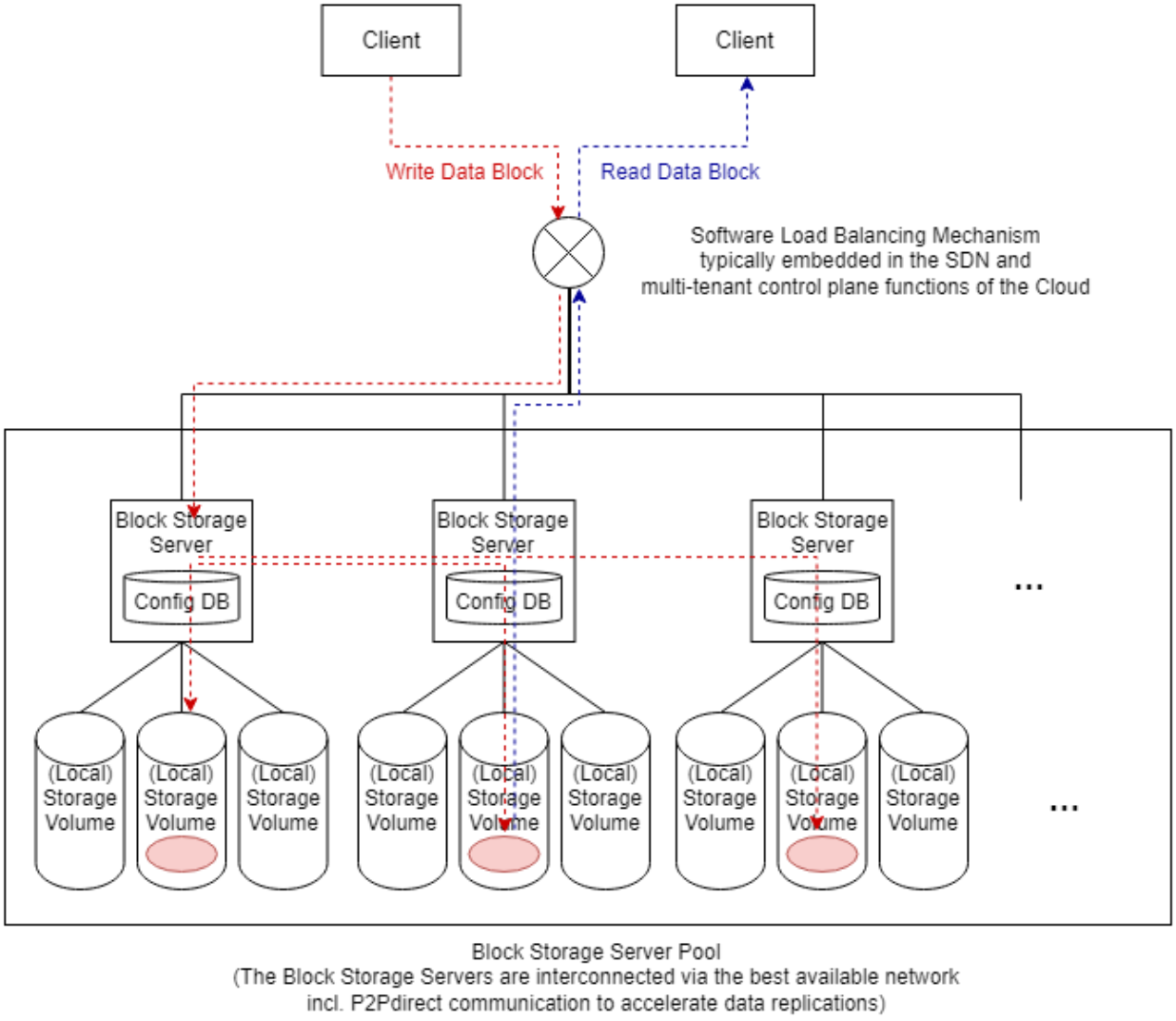

Figure B-1-2, Today's Implementation Model of the Block Storage

The storage I/O capacity, access latency, and available bandwidth of the block storage are critical factors in building a high-performance database system. However, as seen in the above diagram, multiple points slow down the performance, as listed below:

- **Data Replication over limited network resources**

  To guarantee data persistence, the data is copied across multiple block storage servers. This data duplication slows down the write storage access even when the specialized resource pool is used for the Block Storage service because write operations must wait for all servers to finish writing the data to its local storage. In addition, to avoid network congestion and resource contention, it generally applies tight throttling controls for storage access I/O and throughput.

- **The network connection between the block storage and its client (e.g., application server)**

  Due to the large resource pool in the Cloud environment, the network packet should go through multiple network switches; thus, storage access is slowed down.

- **Multi-Tenant Control**

  To build a highly scalable environment, cloud infrastructure typically adopts a mesh-style, clustered network, such as Spine-Leaf-Network architecture. This implies each time when the network packet is exchanged

between two switches, the cloud control plane examines the network packet (header) and determines the physical network path to route the packet. This largely slows down the storage access latency.

Due to such restrictions, Block Storage cannot be used in the same way as Local Storage. For example, storage access latency is much higher, such as one millisecond or more (local storage could provide an order of tens of micro-second or less latency). Storage access IOPS is much lower, such as tens of thousands per 1 TB volume (1 TB PCIe SSD could provide more than hundreds of thousands of IOPS). This is the primary reason DB services in the cloud are much slower than the on-premise DB instance.

B.2. Key-Value Store (KVS)

Existing Technologies Overview

There are various KVS systems already on the market, but these can generally be described as systems with the following characteristics.

- KVS records are identified and accessed by the Primary Key.
- The sharding Key and Distribution Algorithm controls how data is distributed across multiple servers.
- Each record can hold various types of a value flexibly, e.g., a value can be JSON data, to include more detailed data within it.

There is also a type of KVS system that has the following additional features:

- Secondary Indexes against the structured value, to be used for the data query, and joins.
- Dynamic scale-out and scale-in to support workload fluctuations, with a support of the so-call consistent-hash algorithm, which can minimize the data movement during scale-out and scale-in operations.
- Sorting of data records within the server (shard), which accelerates a designated continuous/full scan of data.

However, the KVS solution has its limitation. For example, when compared to a Distributed RDB. KVSs are not very good at several specific applications.

- Strict (i.e., ACID properties) and/or Strong Consistency is difficult to support - some KVS supports the BASE model only, and other KVS may support strong consistency within a shard but not do cross-shard consistency.
    - Note:
        - ACID means “atomicity, consistency, isolation, and durability,” a set of properties that the Distributed RDB should be equipped with.
        - BASE means “basically available, soft state, and eventual consistency.”

Strong consistency means some consistency level between ACID and BASE, e.g., providing ACID for the transactions that occur in a single shard and cover up to N records simultaneously.

· Complex joins against records stored across multiple shards are slow - this is due to 1) an inefficient secondary index mechanism, as you can imagine how the creation of optimal indexes against the data where data structure and statistical distribution are unknown in advance is difficult, and 2) overhead around parsing the “Value” field each time, and extracting data elements within it, etc.

Today’s Implementation Model

A typical implementation model of today’s KVS is shown below.

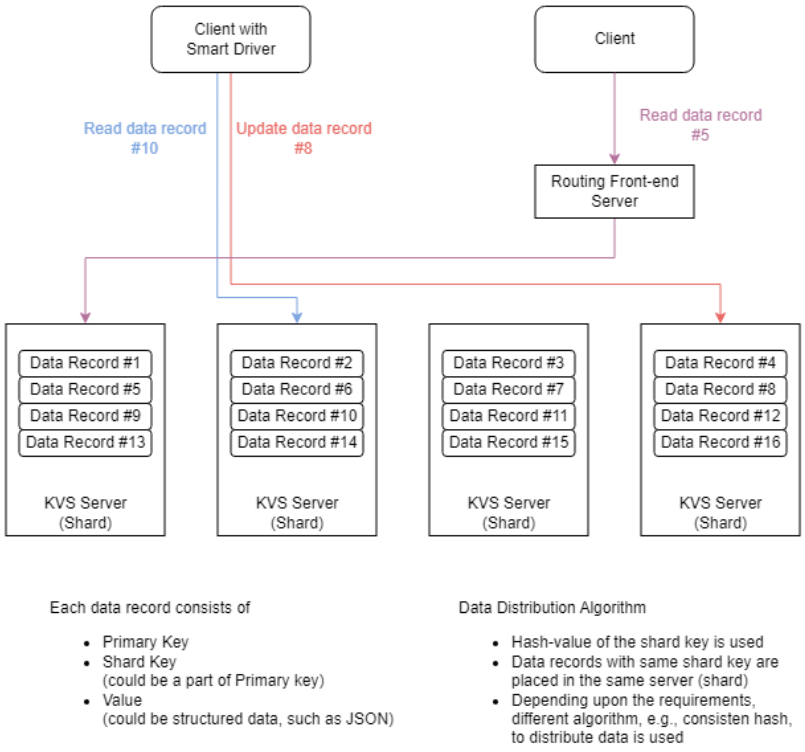

Figure B-2-1, Today's Implementation Models of the Key-Value Store

For complex joins, there are several implementation models in use currently, as described in figure B-2-2.

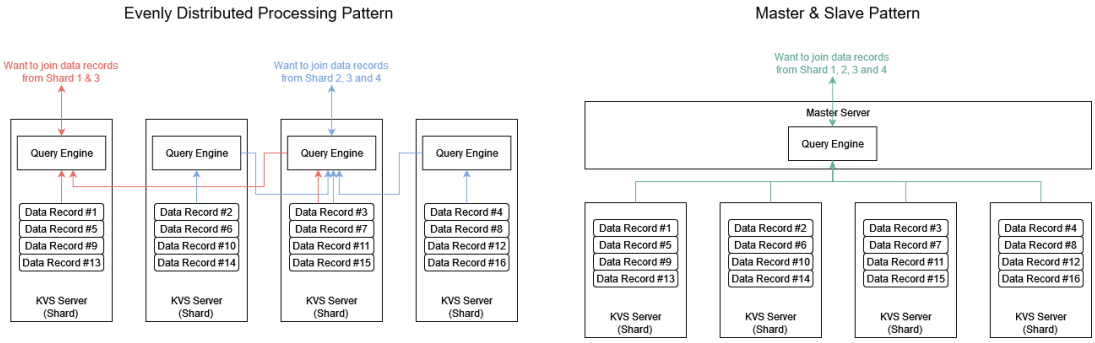

Figure B-2-2, Complex Query Behaviors in the Key-Value Store

Figure B-2-3 below shows how consistent hash algorithms determine data distribution and reduce the amount of data movement in scale up/in operations.

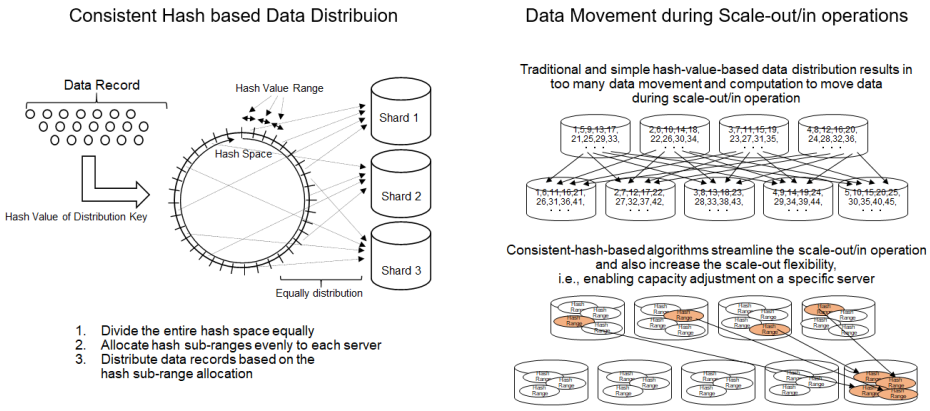

Figure B-2-3, Concept of the Consistent Hash-based Data Distribution

Gap Analysis

The KVS is an excellent technology for processing a large amount of data with outstanding scalability. However, if we look at the implementation model above, we can notice several challenges.

- To decrease the response time, intelligent clients with the smart driver access a KVS Shard where the data resides in a pinpoint manner, by knowing data distribution in advance, for example, by obtaining configuration information from the configuration management server to locate the data before accessing it. However, what happens when scaling-out and scaling-in the KVS system? There will be a delay in delivering configuration change information to the client. Thus, the KVS that received a data access request from the client may need to forward it to the shard where the data resides.
- Too slow, cross-shard joins are problematic when considering use cases like 4D Map. Large data transfers across shards need to be accelerated and optimized no matter what deployment model is chosen. This becomes especially important when the KVS clusters are built across data centers. Another issue exists around spatial and temporal data queries. The secondary indexes in today's KVS are not well-optimized for this type of query. In addition, spatial-temporal analysis is a cumbersome process compared to scalar value processing or matrix operations, so it is necessary to have a suitable mechanism that enables efficient distributed processing on multiple servers.
- Dynamic scale-out / in operations are problematic because it requires data movement. During scale-out / in operations, online processing tends to be slow because many computing resources are used for the data movement process. Some data access requests need to be forwarded to another server, as requested data may already be moved to that server.
- Difficulty in realizing the real-time services, especially when the client is located geographically distant from the cloud data center, as same as for the Distributed RDB.
- Almost all KVSs do not implement granular data access control yet. This means that in KVS, various information is stored as a value within each record designated by the primary key, but there is no mechanism to recognize it and control access to it. In addition, as with RDBs, not many KVSs are implemented with a mechanism that can distinguish multiple application users under a common DB connection. The KVS can flexibly store various information, but there are still many issues in terms of data security.

B.3. Graph Store

Existing Technologies Overview

The Graph Store has existed for decades. In past years, its usage and development have seen significant advancement since its use for social network analysis and web analysis. This type of database stores a graph that is composed of

nodes that are interconnected by edges. Information related to a node or an edge is stored as their properties. With such a flexible data structure, the graph is used to represent and analyze a variety of interests, regardless of the use case.

Today's implementation model

In the Graph Store, data structure is tuned to streamline and speed up the management and exploration of linked data. For example, a KVS may be used to manage a variety of properties in a scalable manner or to embed information on relevant edges in each node record. Another example is to have supplemental information to assist data exploration, such as an index or matrix representing linkage relationships. Figure B-3-1 shows typical examples of such data structure.

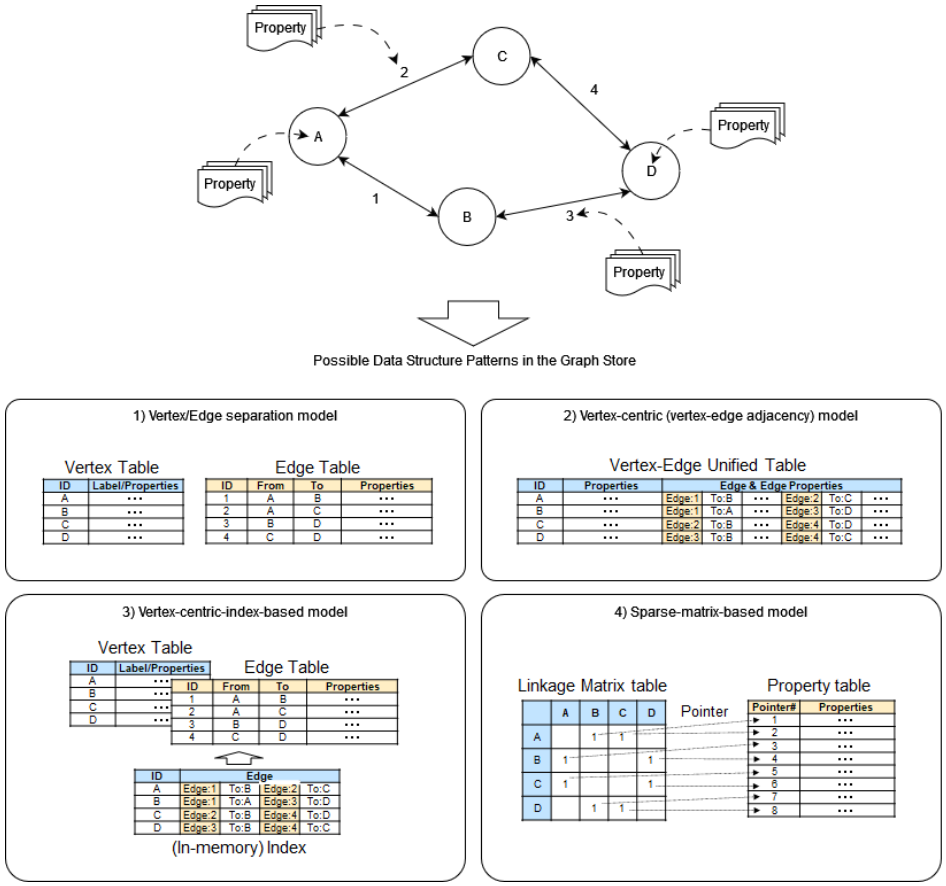

Figure B-3-1, Data Structure Patterns of the Graph Store

To manage a large number of data records, i.e., nodes and edges, multiple servers are used, and data is distributed across them. Data placement is determined by the partition key. For better performance, the partition key is determined to collocate the most relevant records to the same server as much as possible to minimize cross-server links. If the data in the Graph Store needs to be updated frequently, a Graph Store cluster will be built by having two types of servers, one for data management and the other for data analysis. Such an implementation model is described in figure B-3-2.

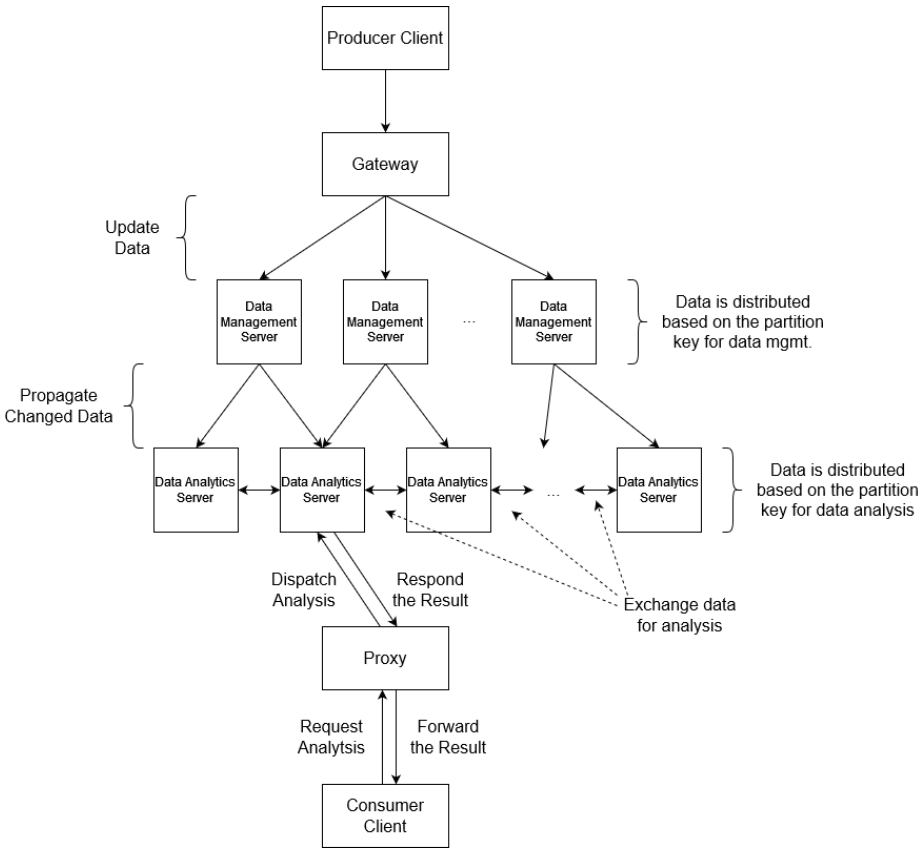

Figure B-3-2, Today's Implementation Models of the Graph Store

Gap Analysis

Data represented in a graph is well suited to represent various concerns in the real world. Therefore, the Graph Store is expected to be a core element of the digital twin, which needs to reflect real-world situations in a real-time manner.

However, as real-world situations change moment by moment, the graph information must be updated frequently. Unfortunately, today's Graph Store is not well-designed for such a dynamicity. This means that even if the implementation model described in Figure 4-3-2 is used, there is still a large time lag before the updated data is available for analysis. This is because, as shown in Figure 4-3-1, each record has a relatively large size, so the storage read/write performance becomes a bottleneck. Of course, the Graph Store can be configured so that data updates and analysis are conducted by the same set of nodes, but then, the throughput of data updates will be more limited, because a lot of CPU resources may be used up for analytics.

Also, to realize the Digital Twin, analytical processes also must be considered. Such analytical processes include identifying current situations, predicting future situations, and determining recommended actions. In this context, the analytical process needs to run a time-series analysis on top of the Graph Store. While properties can hold time-series information, a Graph Store is not well suited to efficiently handle this kind of information yet.

Another issue exists around the real-time analysis. To shorten the response time, analytical processes attached to the Graph Store need to be notified of new information matching the data of interest. Therefore, a mechanism for semantic subscription to the graph may be needed. There are some ways of implementing it with today's Graph Store, such as issuing a message to a Message Broker for every node update. However, this kind of operation needs specific system integration.

The Graph Store can model geographic information and support geo-queries, which are executed by applying geo-filtering on top of semantic query. This is essential in realizing the digital twin. This is because a digital twin is a mirror that reflects the real world in a virtual space, and in that sense it must be able to express a three-dimensional space. However, as the geo-information is treated the same way as any kind of other information in today's Graph Store, the Graph Store cannot differentiate it from any other types of data and optimize the geo-query in most of today's implementation models.

In addition, data security is also an issue. Various information is linked and managed in the Graph Store. However, some of them may contain sensitive information or information that should not be shown depending on the access user's roles and responsibilities. To protect data in the Graph Store, dynamic access controls to vertices and edges are required, but many Graph Stores do not have such a function yet. In addition, as with the Distributed RDB and the KVS, not many Graph Stores are implemented with a mechanism that can distinguish multiple application users under a common DB connection.

Finally, there is an issue around cross-domain management when a Digital Twin of concern includes different domains. Taking the example of the Green Twin, the digital twin representation of a building may be managed by the energy company (thru. smart meters), the local building owner (thru. installed sensors), and the event company that hosts an event in the building (thru. their monitoring system.) at the same time. This means that the graph information is distributed across domains. However, today's Graph Store cannot support such management by disparate stakeholders.

B.4. Message Broker

Existing Technologies Overview

As a large amount of data is continuously transmitted from IoT devices, the Message Broker technologies are used very effectively today. The roles of such Message Broker technologies are ingesting data with high throughput, protecting the data from being lost, and smoothly passing it to the application that uses the data. Examples of such Message Broker technologies are Kafka, Pulsar, and Cloud-based offerings. These technologies are designed to achieve higher-throughput and sequential data writing and reading while providing a unified interface for ingesting data from various devices. However, their main design targets are small payload data such as IoT data and notification messages, thus, the maximum payload size is often limited to something like 1MB, and any messages larger than this threshold needs to be split into chunks before ingested.

Today's Implementation Model

A typical implementation model of today's Message Broker system is shown below. In order to efficiently perform sequential data ingestion and extraction, data is basically stored in order of ingestion, as shown in figure B-4-1. To increase throughput, the system is generally divided into units called partitions. In that case, the order in which data is stored is controlled per partition.

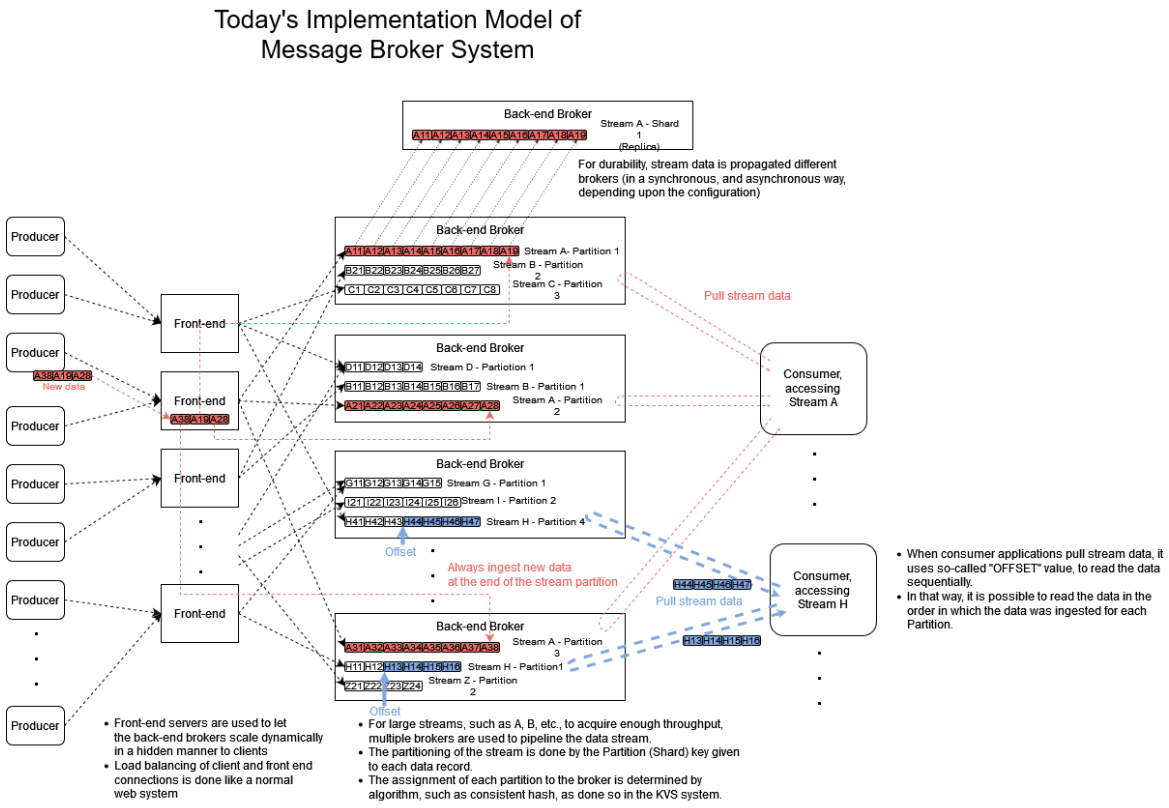

Figure B-4-1, Today's Implementation Models of the Message Broker

Gap Analysis

Today's Message Broker systems are well-designed and can provide high throughput stably. However, there are still some issues to be solved as described below:

- In the use case of mixture data, large-size data will also need to be handled at the same throughput and latency as small data. However, there are few systems in the current Message broker that can efficiently handle data above a specific size. An existing Message Broker system may be able to pipeline over 100K messages every second when the message size is 10 KB, which corresponds to a throughput of 1 GB per second but the achievable throughput will be largely reduced if the message size becomes a few MB or more, because of memory pressure, internal message split process, etc.
- The time lag between when data is ingested and when data becomes available for consumers is not small, e.g., 50 - 100 milliseconds in a Cloud-based typical configuration. This is due to propagating the data so that it is not lost. Especially in the public cloud, it should be noted that data is propagated to adjacent data centers (availability zones) as a fault tolerance measure.
- Only sequential reads are supported. It cannot be used for condition-based record extraction.
- For instance, 4D maps and/or some intelligent applications require condition-based data retrieval because, depending upon the situation, the records of concern are scattered apart in a data stream. To support such a conditional query, today's implementation requires another data management component. The user must ingest data to the Message Broker first, then extract data from there, re-ingest data to another data management, and run the conditional query. Such a series of data processing seems to be overhead for large data streams since these don't create any business value.
- Dynamic scale-out and scale-in are slow. Today's implementation model requires moving from one broker to another during scale-out/in operations while inserting and reading data simultaneously. This is a challenging

problem. Ideally, the vast memory space that pipelines the information needs to be built and shared by multiple brokers to be scaled out / in without moving the data.

- In the IOWN Global Forum use cases, the Message Broker is accessible by the end-user, which requires the Message Broker to have many subscribers. For instance, in a Mobility Service use case, millions of cars send and receive messages throughput Message Brokers. Today's implementation model does not focus on such a scene.
- It is impossible to handle cases where many subscribe, or unsubscribe events are expected to a specific topic. For example, in a Mobility Service use case, if there are hundreds of thousands of cars in a particular area, and many of them go in and out of the area, the topic model of the Message Broker will be challenging to handle messages.
- In the Message Broker, access rights can be managed per partition, but access rights cannot be managed for each message. This may be problematic when the Message Broker mediates very different types of ad-hoc messages because it is not a resource and cost-effective manner to define a specific partition to manage data access rights. (optional)

B.5. Distributed File Storage

Existing Technologies Overview

The Distributed File storage is designed to provide a network file system in a scalable manner, and any instances belonging to the same network can mount it through the NFS, CIFS/SMB, WebDAV, and/or sFTP protocols, etc. Within Distributed File Storage, a very large number of files can be managed hierarchically. Compared to DB services, the Distributed File Storage architecture can be shared among more clients in a more lightweight manner, so it can be used for data processing using many microservices. Also, as long as the connection is maintained, data stored in the Distributed File Storage can be accessed with lower latency and higher throughput than the Object Storage. Therefore, Distributed File Storage is suited for sharing relatively large files among many instances to process them jointly. To protect file data, data at rest and data in transit can be encrypted.

Today's Implementation Model

A typical implementation model of the Distributed File Storage service in today's cloud is shown in figure B-5-1. To be used as a network file share, the frontend server is paced so that the file system provisioned in the Distributed File Storage can be mounted. And through the frontend server, file data is stored on multiple data service servers and attached storage devices. Since the root and other directories are spread across multiple data servers, there is also a configuration management server that manages the overall configuration. In order to protect important files, control functions such as backup to a different AZ at a regular cycle are also performed by that server. In addition, file creation/deletion and data volume are also tracked there for usage-based billing.

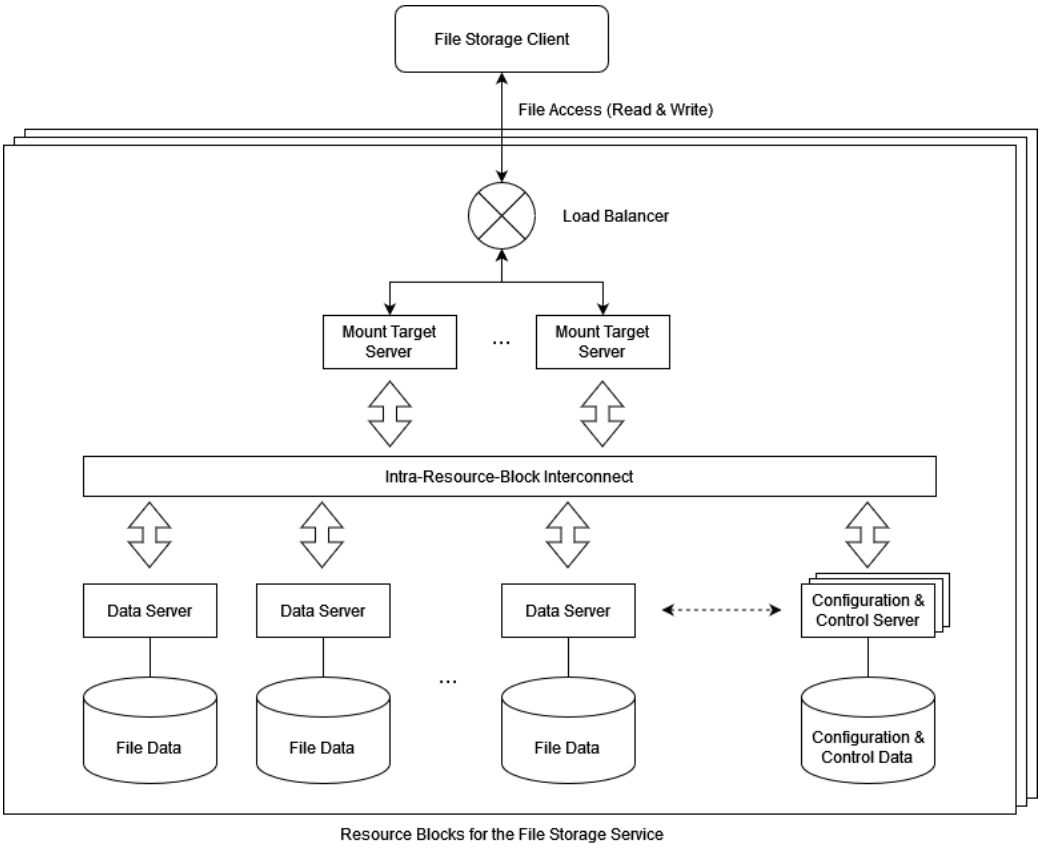

Figure B-5-1, Today's Implementation Models of the Distributed File Storage

Gap Analysis

As today's cloud is a huge resource pool, the network performance between nodes is not so good. As such, the Distributed File Storage is built with certain constraints on performance or scalability. Examples of such constraints are:

- The number of the active frontend server is limited to 1 or 2 per file system provisioned and per availability zone.
    - Thus, the number of concurrent connections is limited, such as up to several tens of thousands, and
    - the read-and-write throughput is also limited, such as up to several Gbps.
- The frontend server(s) and data service servers are placed nearby (within adjacent server racks).
    - Thus, the maximum size of each file system is limited, such as 100 TB.

To realize the future use cases envisioned by IOWN Global Forum, it will be necessary to build a more scalable and faster file system. For instance, it might be desired to let many clients, hundreds of thousands of clients, mount the same file system provisioned by the Distributed File Storage service, and read and write the files at the speed of 100 Gbps or more so that the same data can be jointly and rapidly processed.

Also, to avoid meaningless CPU or GPU waits on the client side due to data transfer, the network connection between clients and the Distributed File Storage shall be faster. Also, for business continuity, there shall be an option to replicate file data across availability zones or regions. However, as the network is not as fast as the interconnects within the resource block, the recovery time objectives provided by Cloud tends to exceed several minutes. Therefore, today's cloud-based file storage cannot be used for mission-critical workloads at the moment.

Lastly, access logging and activity monitoring are becoming more and more necessary as a variety of important information is stored in the file storage. However, today's file storage in the cloud does not implement such logging and monitoring functions enough.

B.6. Object Storage

Existing Technologies Overview

Object Storage is a great technology used frequently within the public cloud. This technology will not provide legacy storage access methods to clients, such as data block-based access as in the local file system and an NFS/CIFS/SMB type of file-based access as in the file server system*1. Such low-layer data access methods/protocols are hidden in the Object Storage, and just the REST APIs are provided to access the data. By doing so, the Object Storage maximizes utilization of the storage resources and compute resources for the data access (i.e., servers to host APIs) and offers the storage service at a very low cost.

*1 By placing the so-called gateway component in front of the Object Storage service, the client can access data stored in Object Storage through file system protocols. However, only stateless APIs such as REST API is included as native APIs for Object Storage.

Today's Implementation Model

A typical implementation model of today's Object Storage is shown in figure B-6-1.

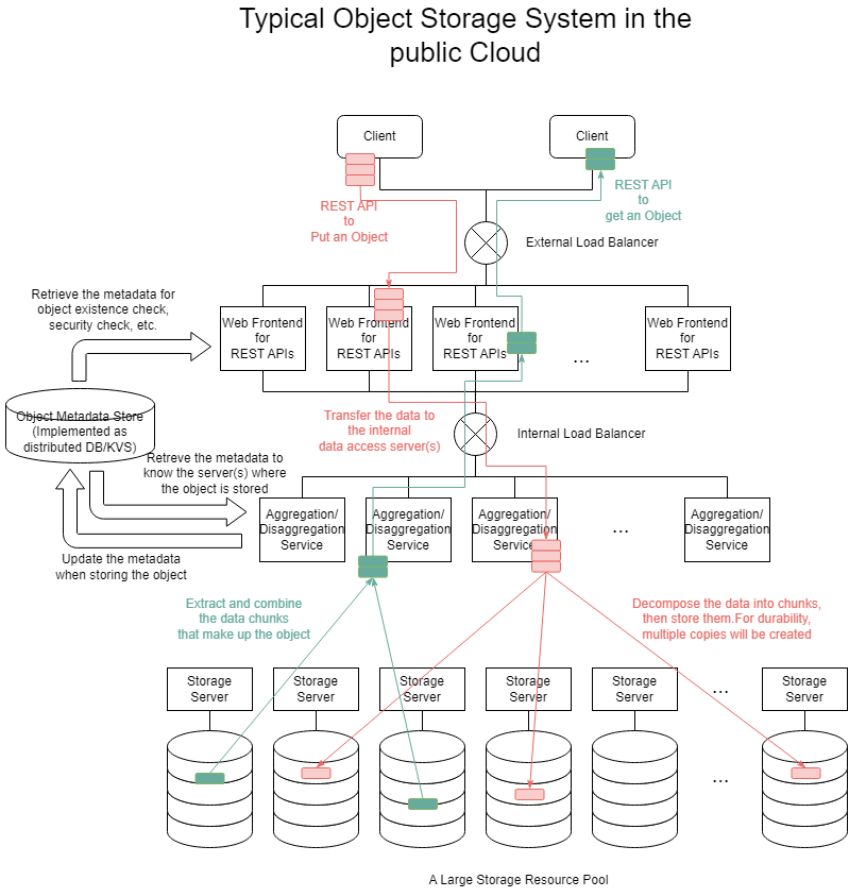

Figure B-6-1, Today's Implementation Models of the Object Storage

Gap Analysis

As shown in the above diagram, the Object Storage system is configured to allow multiple tenants to share the storage resources and scale significantly in the public cloud. However, as a side effect, it is challenging to guarantee stable performance. For instance, it is very difficult to guarantee the response time until the system receives the first byte of requested data and the throughput/bandwidth to continuously extract large amounts of data. As a result, the Object Storage is not used for near-real-time processing today. When used as a data lake infrastructure for big data, it also impacts price/performance ratios and power consumption. Moving data to a server cluster for analysis takes time, while many CPU and GPU resources are left unused. In addition, there is a problem when you put copies of various data into today's cloud-based Object Storage to build a data lake. In many cases, it will end up becoming an unorganized data repository, a so-called "data swamp." Furthermore, in some Object Storage systems, consistency is not guaranteed. This means when the application client overwrote Object X, another client may retrieve the older version of Object X. The delay in data propagation causes this to make multiple copies of object data chunks in the public cloud data center, which is defaulted by the cloud vendor to prevent data loss. At the IOWN Global Forum, the new Object Storage implementations should be studied to enable us to manage and utilize larger amounts of data more efficiently across multiple data centers.

Security is also a very serious issue with deployment of Object Storage. There are many information leaks happening from cloud-based Object Storage implementations today. The reason is that today's cloud-based Object Storage was designed to be exposed on the Internet, making it prone to misconfiguration and hacks. To eradicate this problem, authentication needs to be strengthened and object data to be stored should be encrypted/decrypted on the Object Storage client side rather than on the Object Storage side. Especially for client-side encryption/decryption, it is required to consider not only the data generation application but also the consumer applications, such as the query engine and so on. Therefore, a mechanism to deliver cryptographic keys securely, quickly, and dynamically to those clients need to be built, but such mechanisms are not yet implemented in the cloud today.

Appendix C: IDH Reference Implementation Models

On top of the common architecture structure described in Chapter 5, this section explains additional details for each IDH service, such as unique structural requirements and special designs required to fulfill these.

C.1. Reference Implementation Models for each Basic Service Class

In this section, IDH reference implementation models for each basic service class are explained.

C.1.1. Distributed Relational Database (Distributed RDB)

The basic structure of a Distributed RDB is shown in figure C-1-1-1, which consists of four layers; the smart client, a gateway (optional), DB server, and storage, each of which is interconnected via a network.

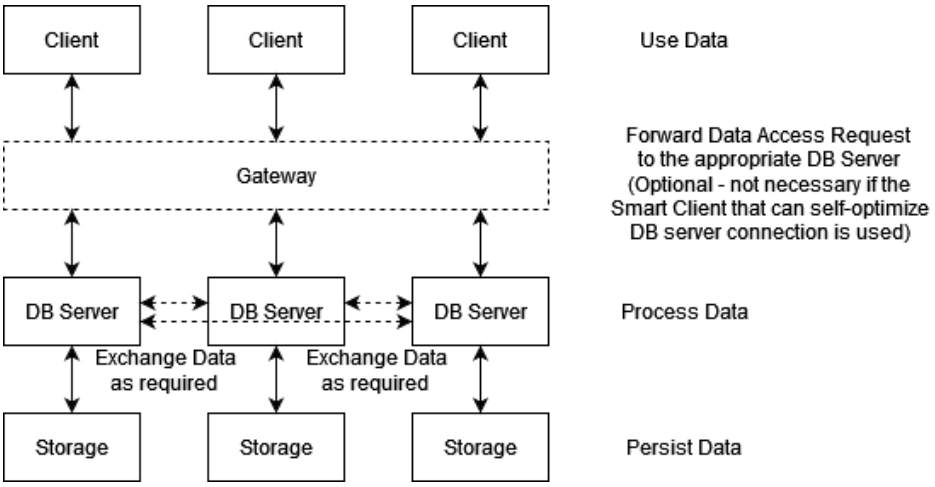

Figure C-1-1-1, Basic Structure of the Distributed Relational DB

As described in section 5.1, IOWN OpenAPN and IOWN DCI will accelerate communications between components that make up the Distributed RDB and data extraction from storage to improve its performance. In addition to these, some ingenuity will be added to the IDH Distributed RDB to accelerate its characteristic functionality, such as updating large amounts of data while maintaining consistency, high-speed data query from various perspectives, etc. Below is a list of the components and network connections that make up the IDH Distributed RDB, with explanations of ingenuity implemented, if any.

- **Smart Client**

  As described in section 5.1, the clients will become smart to send the data access requests to the most appropriate DB server where more request data exists in the cache. Given the IOWN Global Forum use case, we'll find very low-latency data management requirements, such as digital twin infrastructure representing real-world situations in virtual space within tens of milliseconds or less. To achieve such low latency data processing, extra hops that require the software-defined controls in the communication path should be eliminated, which requires the client to be smart. To assist such a smart client, the IDH Distributed RDB will deliver the data placement knowledge to all clients and ensure the information inside the clients is always up to date. The smart client also measures the latency required to communicate with each DB server to prioritize the DB servers to connect to when data is replicated to multiple DB servers.

- **Gateway**

  An endpoint capability to securely connect various clients of various tenants to the IDH Distributed RDB on a shared environment will be embedded in the gateway component if the IDH Distributed RDB is configured as multi-tenant. With this feature, the tenant to which the access request belongs is immediately determined, and after applying appropriate security checks, connection to the appropriate DB instance on the same physical server is permitted. In today's cloud, this is implemented as software-defined controls against the tag attached to each network packet, which is one factor that causes a delay in accessing DB services. As a result, a typical DB service response time becomes more than a few milliseconds. In IOWN, this delay will be reduced by leveraging the power of the IOWN OpenAPN architecture and the IOWN DCI architecture.

- **DB Server**

  In the DB server, the DB engine will become more intelligent to optimize the query optimization plan by considering the data placement, available network bandwidth, and latency for all server combinations in the cluster. It will optimize the overall data processing plan, such as which DB server to collect data from the storage, process data on which server, transfer data processing requests or data when cross-server processing is required, etc.

- **Storage**

  Since tabular data is stored in the storage, several accelerators suitable to process it will be embedded, for example, speeding up encryption/decryption, performing column-oriented compression, creating a summary index for each data block to speed up full-scan queries, etc.

- **Smart Client - Gateway - DB Server Connection**

- **Inter-DB Server Connection**

  The DB server cluster may be configured to be geo-redundant, as is done for the high-availability configuration in today’s cloud-based implementation models. Even in such cases, the flexibility implemented in the IOWN DCI Gateway provides a faster network to connect DB servers between different data centers. This makes it possible to run DB Services that require high reliability faster than ever.

- **DB Server - Storage Connection**

Figure C-1-1-2 shows images of the IDH Distributed RDB reference implementation models with such ingenuity described above.

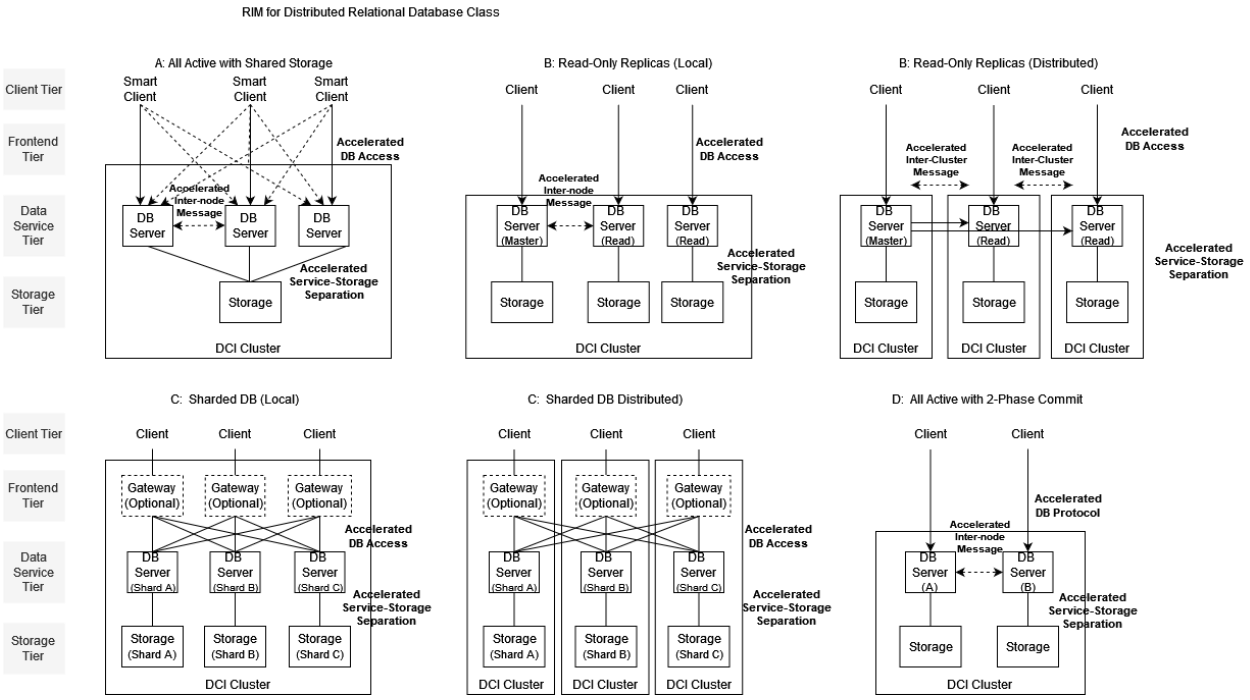

Figure C-1-1-2, Reference Implementation Models of the IDH Distributed Relational DB

Figure C-1-1-3 below shows images of the IDH Block Storage System reference implementation models that can be used to build an IDH Distributed RDB, an IDH KVS, an IDH Graph Store, and so on. As shown in the figure, there are multiple topology options that are selected from the viewpoint of performance and scalability while using the IOWN OpenAPN to accelerate communication as a common factor. Characteristics of each option are described in Table C-1-1-1.

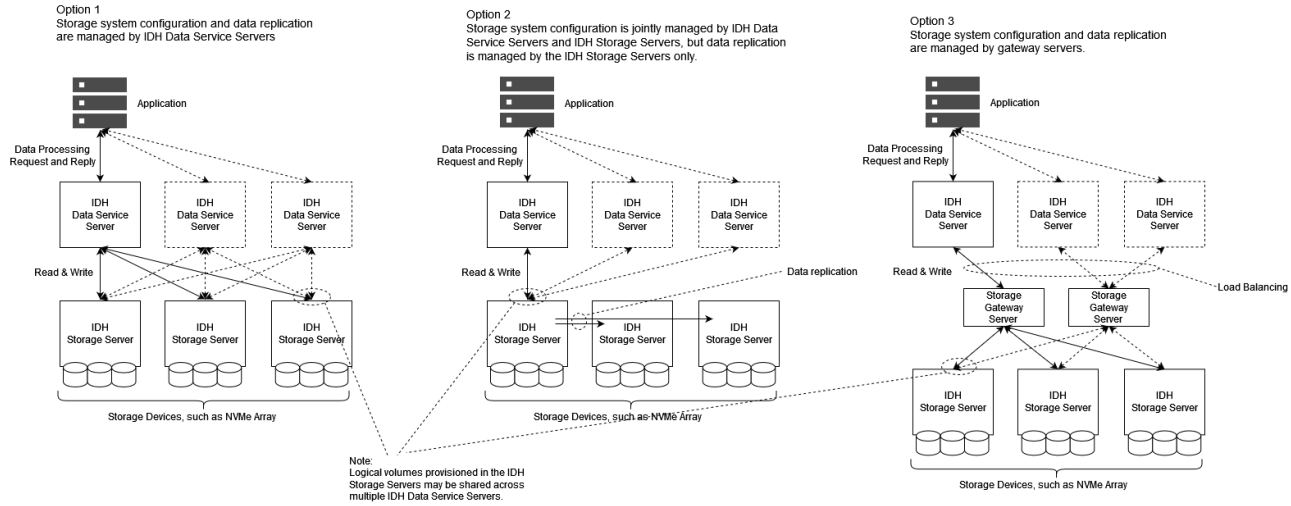

Figure C-1-1-3, IDH Block Storage Reference Implementation Model Patterns

Table C-1-1-1, Characteristics of each IDH Block Storage Reference Implementation Model Pattern

	OPTION 1	OPTION 2	OPTION 3
Read & Write Latency	Very Good (e.g., <200 µsec)	Good (e.g., <1msec)	Not Bad (e.g., <1.5msec)
Maximum Throughput (IOPS)	Very Good (e.g., 10 million IOPS)	Not Bad (e.g., 0.5 million IOPS)	Good (e.g., 1 million IOPS)
Maximum Volume Size	Very Good (e.g., >100 TB)	Good (e.g., 1 million IOPS)	Very Good (e.g., >100 TB)
Performance Configurability	Fixed (IOPS is proportional to storage size)	Fixed (IOPS is proportional to storage size)	Configurable (Flexible IOPS to Volume Size ratio)
Cluster Size	Limited (e.g., up to tens of server racks)	Flexible (e.g., >1,000 server racks)	Flexible (e.g., >1,000 server racks)
Control Plane	Embedded in the data servers	Distributed across the storage servers	Semi-distributed across the gateway servers

C.1.2. Key-Value Store (KVS)

The basic structure of the Key-Value Store (KVS) is shown in figure C-1-2-1, which consists of four layers; clients, routing frontend (optional), KVS servers, and storages, each of which is interconnected via a network.

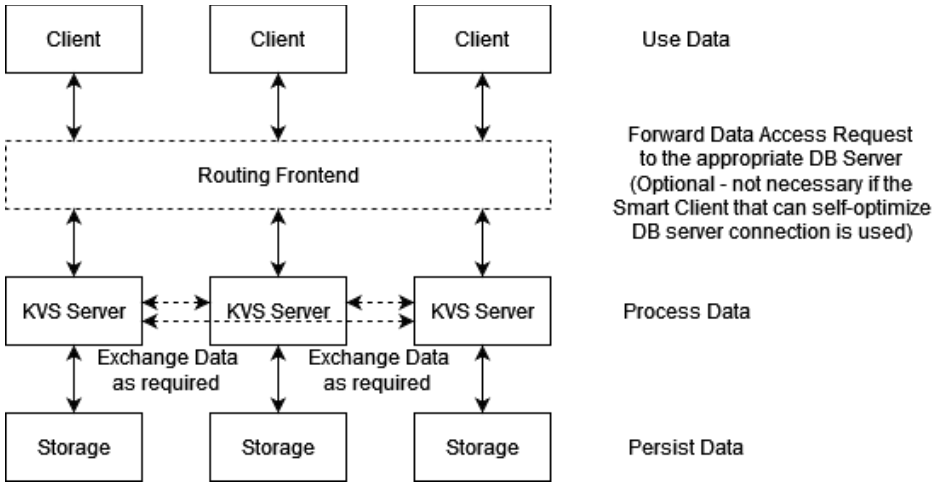

Figure C-1-2-1, Basic Structure of the Key-Value Store

The IDH KVS performance is accelerated through mechanisms described in section 5.1 while ensuring the scalability of key-based simple access. On top of that, to support various data query capabilities and to increase flexibility when data distribution and/or workload distribution changes, additional ingenuity will be implemented. Below is a list of the components and network connections that make up the IDH KVS and explanations of ingenuity, if any.

- **Smart Client**

Like the IDH Distributed RDB case, the clients will become smart to send the data access requests to the most appropriate KVS server where (most of) the request data exists in the cache.

- **Frontend**
- **Key-Value Store (KVS) Server**

  The IDH KVS server will be decoupled from the storage. This means improvements to the current situation in which the KVS servers rely on the local storage and are coupled with storages tightly. With such a configuration, if the distribution of data and workloads were biased, it would have been necessary to change the number of servers and relocate the data to level out the data and workload. However, such operations are difficult, even if a good algorithm such as consistent hash is used. In the IDH KVS, it is only required to dynamically adjust the data allocation from storage to KVS servers, depending on the amount of data and the workloads. The high-speed IOWN OpenAPN network makes this possible by allowing remote storage to be used as if it were local storage.

- **Storage**

  In the IDH KVS, the storage layer becomes a set of storage servers that provides block access to data. In such a storage server, flexible volume management will be implemented so that data blocks stored in the storage layer can be dynamically assigned to any KVS server as required, such as for data redistribution controlled by the consistent hash algorithm. Such an advanced storage server enables a more flexible, scalable, and high-performance KVS service.

- **Smart Client - Frontend Server - KVS Server Connection**
- **Inter-KVS Server Connection**
- **KVS Server - Storage Connection**

Figure C-1-2-2 shows images of the IDH KVS reference implementation model with such ingenuity described above.

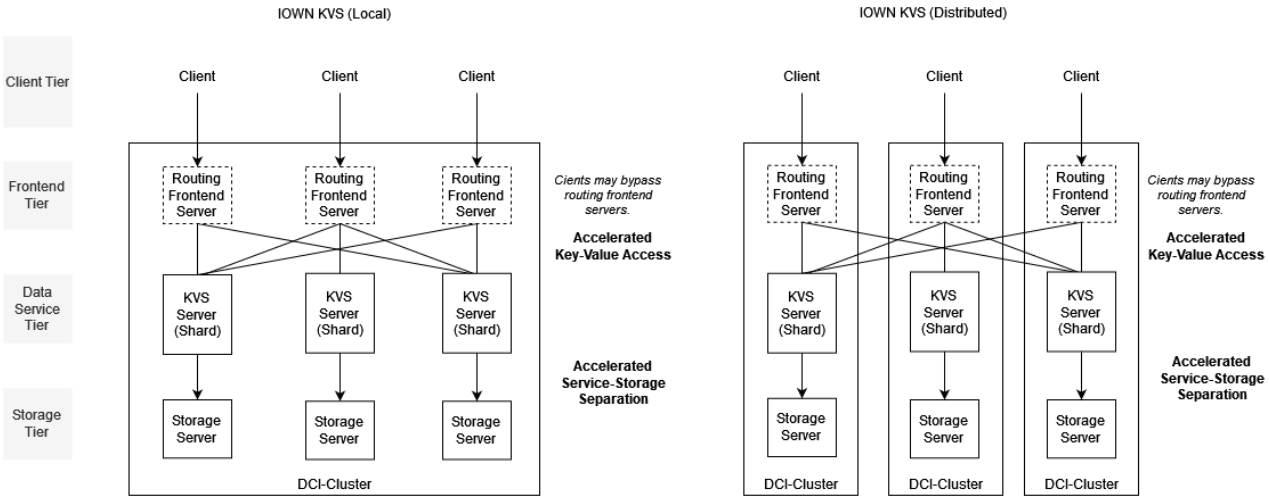

Figure C-1-2-2, Reference Implementation Models of the IDH Key-Value Store

C.1.3. Graph Store

The IDH Graph Store will consist of four layers; clients, gateway, Graph Store servers, and storage, each of which is interconnected via an ultra-fast network, too.

To develop an advanced Graph Store, the gaps described in section 4.3 must be addressed. To effectively support the massive updates of the graph data, faster storage is required. To realize real-time graph analysis, the multi-version concurrency control mechanism will be introduced, and data updates and analysis will be performed on the same set of servers. To let multiple servers collaboratively and efficiently search and analyze large graph data at higher speed,

the index of linked data may be managed in shared high-speed memory. To protect sensitive and private information, node, edge and property-level access controls will be introduced. There will also be a framework for achieving secure integration of multi-party graph data.

Below is a list of the components and network connections that make up such an IDH Graph Store, with explanations of technical ingenuity if any.

- **Client**
- **Gateway**

  In the IDH Graph Store, the gateway considers the placement of data and the workload status on each Graph Store server more precisely. This is because the execution time of a graph search and analysis job can be very long. Therefore, while trying to publish jobs to servers where more data is cached, jobs are dispatched to the appropriate server set in consideration of expected CPU cycle usage.
- **Graph Store Server**

  In a typical today's implementation model, two types of servers, one for data management and the other for data analytics are used, and the graph data is replicated between them. In the IOWN Global Forum reference implementation model, these two server roles are combined, and analytical processes can be executed within the Graph Store server, and these processes running on top of assigned CPUs and accelerators can directly analyze the data. Therefore, unnecessary data replication is abolished, and thus the required system resources are reduced, and processing speed is increased.

  The IDH Graph Store implementation model also speeds up communication between Graph Store servers, which speeds up queries that search graph data across servers. Also, when updating the graph data, a new version of the record is created. This eliminates the need to block read access for analytics during data updates to guarantee data consistency. Also, when the data update is completed, the IDH Graph Store can trigger subsequent processing if updated data matches the predefined conditions so that a real-time analysis and response can be made.

  In addition, graph index and data are managed in a shared in-memory pool or shared storage connected through an optical network. With such a shared index, multiple servers can jointly explore and analyze graph data at high speed. With such shared storage, required data can be loaded to each server as required.

  Also, the IDH Graph Store server can take security into consideration. It controls who can access which vertices, edges, and properties.

  Lastly, the IDH Graph Store server provides a layer for semantic integration with other Graph data. This is because the digital twin type of use cases requires open and semantic data integration among multiple stakeholders. The IDH Graph Store server can accelerate this by utilizing the IOWN OpenAPN and various accelerators provisioned by IOWN DCI.
- **Storage**

  As discussed in Section 4.3, graph data can have large record sizes. The IDH storage is configured to support high-speed write access to such large records by applying the IOWN OpenAPN to the data replication mechanism at the storage layer.

  The IDH Graph Store also includes a shared in-memory pool connected through an optical network to manage indexes to accelerate huge graph analysis. With this, it becomes possible to achieve consistent and high-speed analysis for huge graph data.

  Lastly, the encoding and decoding of graph data may be accelerated by DPU/IPU attached to the storage server. This is because, as each record of the graph tends to contain various linkages and properties information, the encoding/decoding mechanism is important for the performance of the IDH Graph Store, and this can be improved by applying accelerators.
- **Smart Client - Gateway - Graph Store Server Connection**

  For faster data transfer, these are connected through the IOWN OpenAPN.

- **Inter-Graph Store Server Connection**

  To speed up graph search and analysis by massively parallel processing with hundreds of system resources, it is important that those system resources are connected by a low-latency network. The IOWN DCI provides such a high-speed network, and the IDH Graph Store server cluster is configured on top of it.

- **Graph Store Server - Storage Connection**

  For faster data transfer, the Graph Store server and storage server is connected through the IOWN OpenAPN.

Figure C-1-3-1 shows images of the IDH Graph Store reference implementation model with such ingenuity described above.

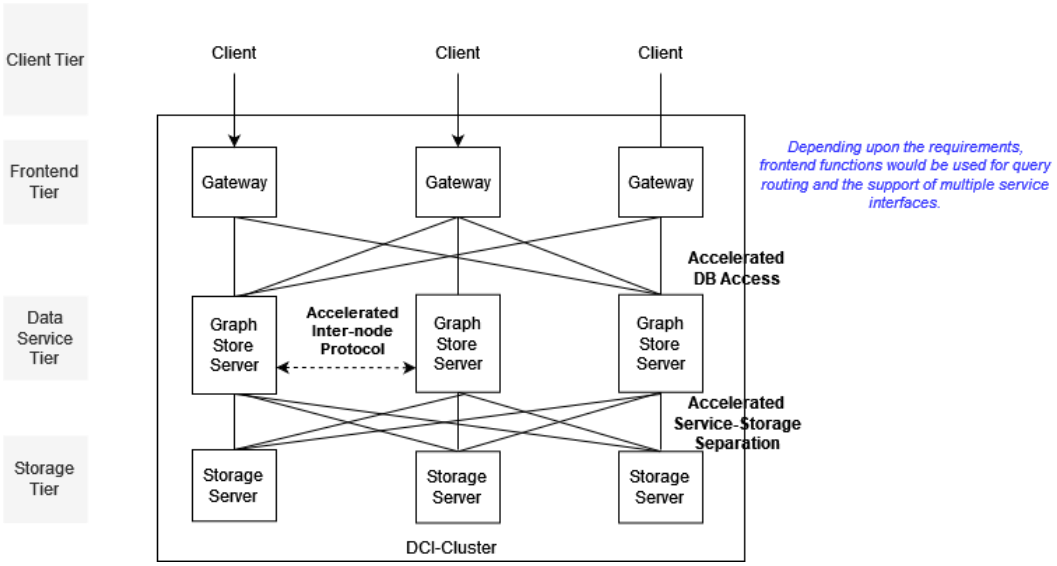

Figure C-1-3-1, Reference Implementation Models of the IOWN IDH Graph Store

C.1.4. Message Broker

In a Message Broker system, data of various sizes, types, and formats are ingested from various devices and passed to the subsequent consumer applications in which data is processed in different ways. To support not only sequential and delayed queuing, but also real-time event integration and query-based flexible data extraction, the Message Broker is comprised of two types of components, “data broker” and ”event broker”, in the IDH reference implementation model. The data broker is designed to queue data that is too large for immediate processing in other systems, so that data can be processed sequentially or in a batch manner. The event broker is designed to pipeline small messages as well as the metadata of large messages, so that processes at the subsequent applications can be triggered in an event-driven manner with low latency. It also supports queries to know which data is ingested and kept in the Message Broker.

Therefore, the IDH Message Broker can be characterized as:

1. The use case of mixture data: It can handle both small-size data and extensive size data.
2. High throughput: It achieves high throughput regardless of the size, type, and format of data.
3. Low latency: It notifies data generation to clients asynchronously to avoid the bottleneck of data transmission.

Below is a list of the components and network connections that make up the IDH Message Broker and explanations of the ingenuity implemented.

- **Client**

The client library to utilize features of the IDH Message Broker is installed in the client. The client library provides Kafka/Pulsar-feel APIs to interact with a data broker and an event broker, including:

- Topic management such as Create, Delete, and List topics.
- Partition management such as Create, Delete, and List partitions.
- Publish messages to the Message Broker
- Subscribe to the topics/partitions so that message arrival is notified immediately
- Queries message metadata to get an overview of what data is being ingested
- Retrieve messages sequentially or conditionally from the Message Broker

One of the main distinctions between this reference implementation model and existing products is that the client library automatically distributes messages across data brokers and event brokers for the lowest latency and the highest throughput when transferring messages.

- **Frontend**

  The frontend is an optional component for high availability and workload distribution. It provides a unified endpoint for the end user and transfers messages to one or more brokers by considering message size. Without having it, the high availability requirement is not well supported, as all messages will be sent to the data broker. Since the event broker will be populated from the data broker, latency is increased.

- **Data Broker and High Throughput Storage**

  The data broker and the high throughput storage are designed to receive and persistent large-scale data streams at very high throughput. To achieve this, any incoming stream can be treated as a byte stream to achieve the best performance. If configured, it also ingests the metadata of received data to the event broker.

- **Event Broker and Real-time Storage**

  The event broker and the real-time storage are responsible for routing and storing small messages as a single object at low latency. To minimize the latency, rapid storage, such as persistent memory is used to store data. If configured, it notifies subscribed clients about message arrival in an event-driven manner to enable real-time processing. It also provides query capability to enable not only sequential reads but also conditional reads. In existing Message Brokers, each stream is composed of resources with a fixed throughput, so one stream often contains multiple sensor data. Since only pull-type sequential reads were supported, there are issues with real-time delivery and processing efficiency. The event broker will eliminate these issues.

Figure C-1-4-1 shows images of the IDH Message Broker reference implementation model with such ingenuity described above.

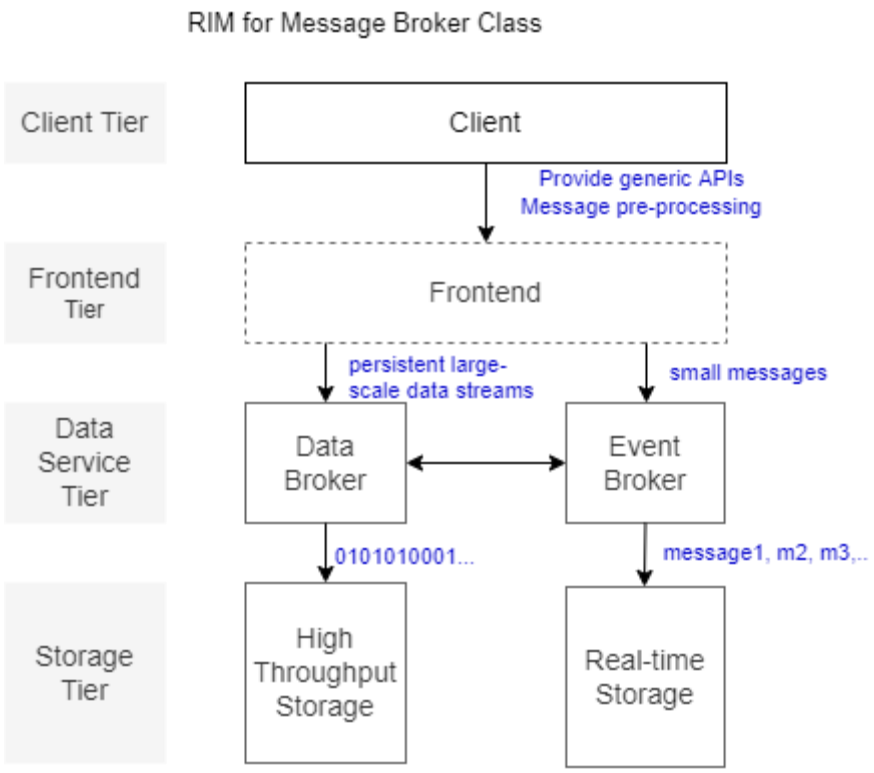

Figure C-1-4-1, Reference Implementation Models of the IOWN IDH Message Broker

C.1.5. Distributed File Storage

To eliminate the gaps identified in Chapter 4, the IDH Distributed File Storage shall be designed as follows:

1. Allows having a large number of frontend servers for larger throughput and more concurrent client connections with fast connectivity between frontend servers and data service Servers
2. Allows storage capacity to be expanded freely with fast connectivity between data service servers and frontend server
3. Reduce latency of file access by speeding up the connection between the client and the frontend server, and between the frontend server and the data service server

The basic structure of the Distributed File Storage determined as such is shown in figure C-1-5-1, which consists of four layers; clients, frontend servers, data service servers, and storage, each of which is interconnected via a network.

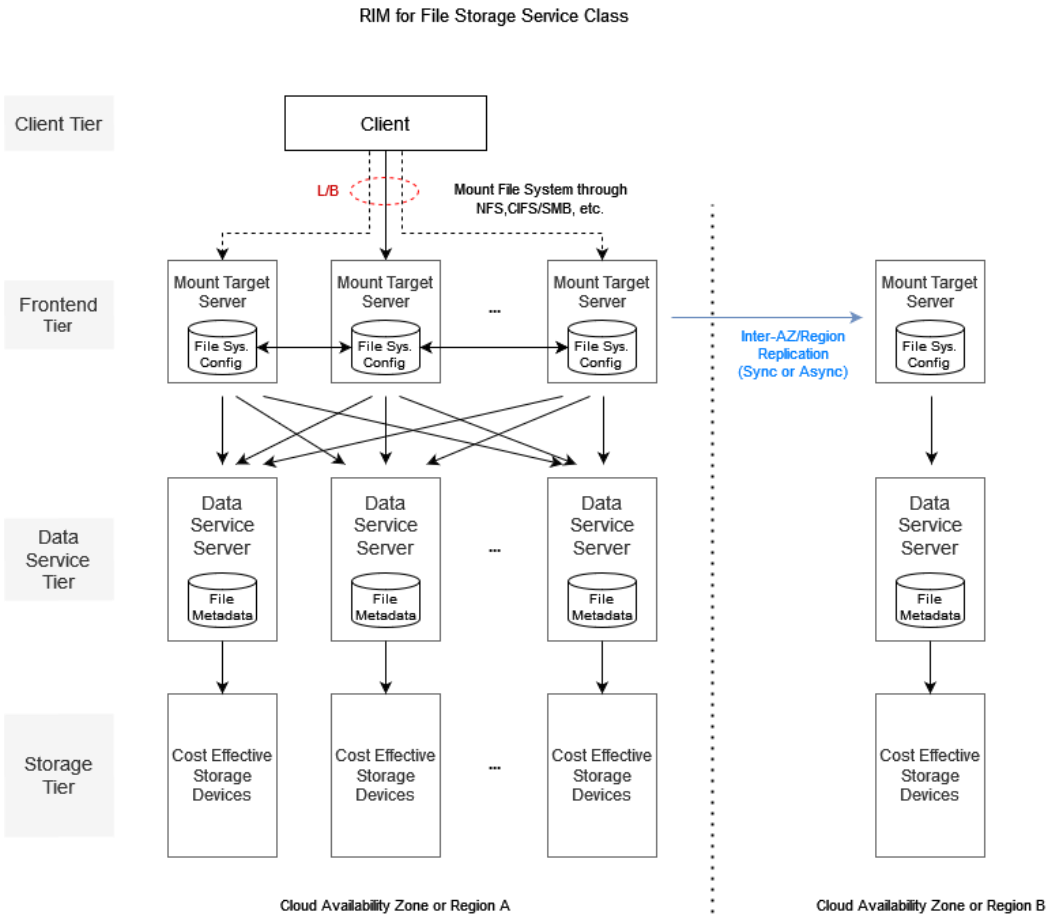

Figure C-1-5-1, Reference Implementation Model of the IDH Distributed File Storage

Below is a list of the components and network connections that make up the IDH Distributed File Storage and explanations of ingenuity implemented, if any.

- **Client**

  The IDH Distributed File Storage client is software that reads and writes the file data stored in a file system provisioned by the IDH Distributed File Storage system. The following features must be supported for the client:

  - The file system is mounted as a directory by the client through file storage API, such as NFS.
  - File access latency (time to the first byte) shall be sub-microsecond.
  - Hundreds of thousands of clients can mount the same file storage.

- **Frontend**

  The frontend server acts as a mounting target of the provisioned file system in the file storage. The following features must be supported for the frontend server:

  - Manage the file system and the directory structure provisioned
    - A single file system will be distributed and managed by multiple data service servers.
    - Moving files to level the workload across the data service server by considering access frequency and/or freshness of the files
  - Provide file access security

        - User-level authentication such as Kerberos
        - Host-level access control such as IP address restriction and/or wavelength-based path restriction
        - Audit log of the selected actions
        - Data-in-transit and data-at-rest encryption
            - User-specified encryption key management per filesystem basis with key rotation support based on the envelope encryption technique.
            - Hardware Security Module (HSM) is built into the accelerator to securely protect encryption keys
    - Provide non-functional features
        - Scale-out/in by adding/deleting frontend servers without sacrificing performance.
            - To provide consistent access, the file system configuration information such as directory hierarchy is synchronized across multiple frontend servers assigned to the tenant.
        - Scale-up/down by adjusting CPU and memory resource allocation to each frontend server so that resource usage becomes efficient.
        - Replicating file data beyond the cloud availability zone and/or the region

- **Data Service Server**

  The data service servers manage the file data in a distributed manner for multiple tenants. The following features must be supported for the data service server:
    - Manage file metadata, such as access rights, properties, locking status, etc.
    - Store and retrieve the file data to/from the attached storage devices.
    - Detect identical files or data blocks and merge them together to save the storage capacity.
    - Record the performance-relevant metrics.
    - Validate all file access requests are within a tenant's scope of concern.
- **Storage**

  The storage stores the actual data of the file. The following features must be supported for the storage:
    - Support various devices such as SSD, HDD, and Tape to store the data depending upon the file storage type.
    - Can be configured with simple and many directly-attached devices so that it can earn capacity at a lower cost are used, as the amount of CPU and memory resources required is not so large compared to the amount of data in the file storage system.
- **Client - Frontend Connection and Frontend – Data Service Server connection**

  To accelerate and increase the flexibility of the file access, both the connection between the client and the frontend server and the connection between the frontend server and the data service server shall be scalable and established by the IOWN OpenAPN and the IOWN DCI. The following features must be supported for these connections:
    - Massive concurrent connection support, i.e., tens of thousands or millions of clients can mount the same file system simultaneously
    - Scalable frontend, i.e., the number of frontend servers, can be changed online to match the number of clients mounting the file system
- **Data Service Server – Storage Connection**

The storage devices are connected as a local resource to the data service server, as there is no need to decouple the data service server and storage. This is because the workload on each data service server is primarily based on file access I/O, and there is little variation in workload between servers if they store almost the same amount of data. TBD

C.1.6. Object Storage

The basic structure of the Object Storage is shown in figure C-1-6-1, which consists of four layers; clients, web frontend servers, data service servers, and storage, each of which is interconnected via a network.

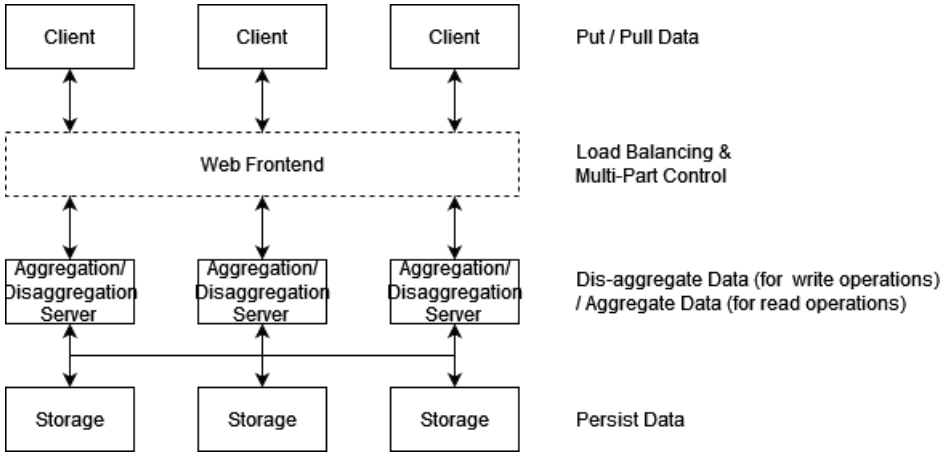

Figure C-1-6-1, Basic Structure of the IDH Object Storage

With the explosion of data volume, the Object Storage has become the core of the data lake. Therefore, some ingenuity will be added to improve the performance of the IDH Object Storage. Below is a list of the components and network connections that make up the IDH Object Storage and explanations of ingenuity implemented, if any.

- **Client**

  On the client side, only Put and Get REST APIs are used to access the data stored in Object Storage in today's implementation model. In the IDH model, these access methods are extended by the client library. For instance, a client library that can:

  - put and get objects through REST APIs, including AWS S3-compatible ones
  - accommodate other protocols such as CIFS/SMB,
  - efficiently upload and/or download data based on UDP and/or RDMA protocols, as well as multi-part access controls, on top of the IOWN OpenAPN
  - designate query conditions to retrieve only the data chunks that the client needs, such as blocks in the parquet file and/or video segments in the video file, etc.

  These new features are made possible by improving the Object Storage implementation model as described in this section.

- **Web Frontend**

  The roles of web frontend are 1) to interact with clients, including authenticate and authorization, confirmation of the existence of the requested object, enabling access to the object data for its clients by providing a set of general APIs compatible with current cloud services and existing on-premise storage systems, coordinating multipart upload and download, etc., 2) managing metadata of the object and also to provide internal load balancing controls for performance and availability. In today's implementation model for the Object Storage, the ratio of computing resources to storage resources is fixed. Therefore, the web frontend only provides very limited APIs, such as Put and Get. With the IOWN DCI architecture, the resource pool can be configured more flexibly and dynamically. Therefore, other types of functionality can be

added to the web frontend, such as the file system protocols to streamline the data access in the Object Storage without requiring a gateway server needed in today's implementation model.

- **Aggregation / Dis-aggregation Server**

  The roles of the Aggregation / Dis-aggregation server are 1) to dis-aggregate the received data into multiple blocks before storing them when the client uploads an object to the Object Storage, and 2) to aggregate the several blocks that make up the object to pass it to the client when the client downloads an object from the Object Storage. In today's implementation model, splitting this data into parts is simply based on data size. In the IDH implementation model, splitting into parts will become more intelligent by considering the content. For instance, the parts for the parquet file will be determined by the parquet's internal blocks. The parts for the video file will be determined by the video file's internal segmentation, such as a chapter or key-frame-based segmentation. In addition, index information for these chunks of data will be created, such as minimum-maximum values for each column stored in the parquet block and collection of subtitle texts, start and end timestamps, etc., for each video segment. By having such a structure and allowing the client to specify the data retrieval conditions, only the portion of the data that the client needs can be extracted from the Object Storage efficiently and at high speed. This resolves inefficiencies we frequently see in Object Storage usage today. To retrieve only a small portion of the object, the entire object needs to be retrieved from the Object Storage, which is time-consuming and costly for its users.

- **Storage Tier**

  The role of the storage tier is to persist the object chunks created above and internal index information securely. To guarantee data persistence, data is replicated three times among different servers typically located some distance apart. The IOWN DCI technology for the inter-cluster network will accelerate this data replication. In addition, to protect the data-at-rest from any unauthorized access, the data will be encrypted. The IOWN DCI can also offload such tasks from the CPU to the storage, e.g., Smart NIC installed there, and encrypt the data with a customer-specified encryption key.

  In addition, the storage tier may filter portions of the object and transfer only the required elements to the data service server. This behavior occurs when the client designates the content-aware filtering conditions, and the required compute resource is available at the storage.

- **Client - Web Frontend Connection**

  If a particular client application uploads and/or downloads a large amount of object data to/from the Object Storage continuously, such a data transfer will be accelerated by the IOWN OpenAPN and the IOWN DCI.

- **Web Frontend - Aggregation / Dis-aggregation Server - Storage Connection**

Figure C-1-6-2 shows images of the IDH Object Storage reference implementation model with such ingenuity described above.

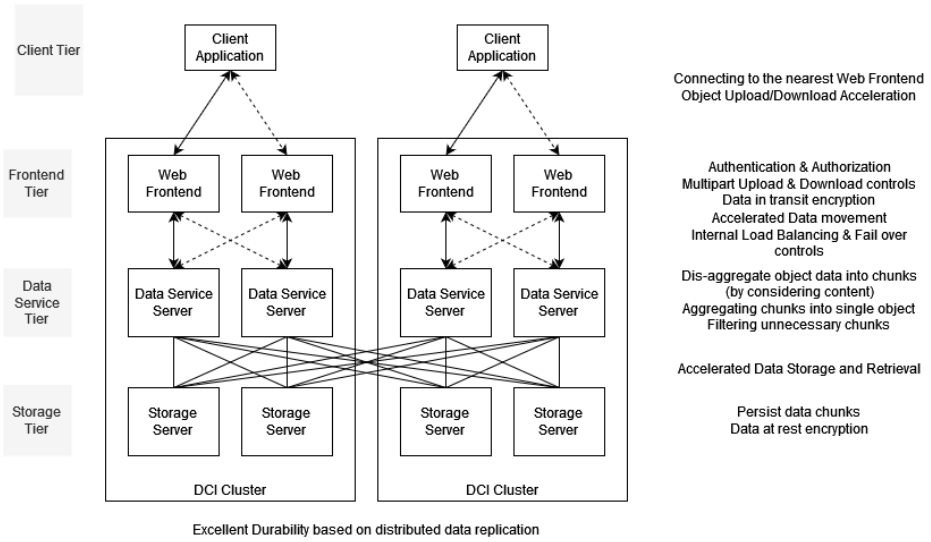

Figure C-1-6-2, Reference Implementation Model of the IDH Object Storage

C.2. Reference Implementation Models for each Applied Service Class

C.2.1. Context Broker

The Context Broker has a distributed nature since it intermediates the access of context data between the clients and storage or other Context Broker instances and context sources. Context data is handled by different domains that collaborate with each other. However, the context consumer needs to issue a request to only a single Context Broker that is responsible for provisioning data from its own managed context and external context sources or domains.

Figure C-2-1-1 shows the IDH Context Broker reference implementation model, with the interaction of subcomponents of the Context Broker divided into the client, frontend, data service, and storage tiers. The picture below is a meshed configuration; however, there might be a hierarchical configuration or a hybrid configuration.

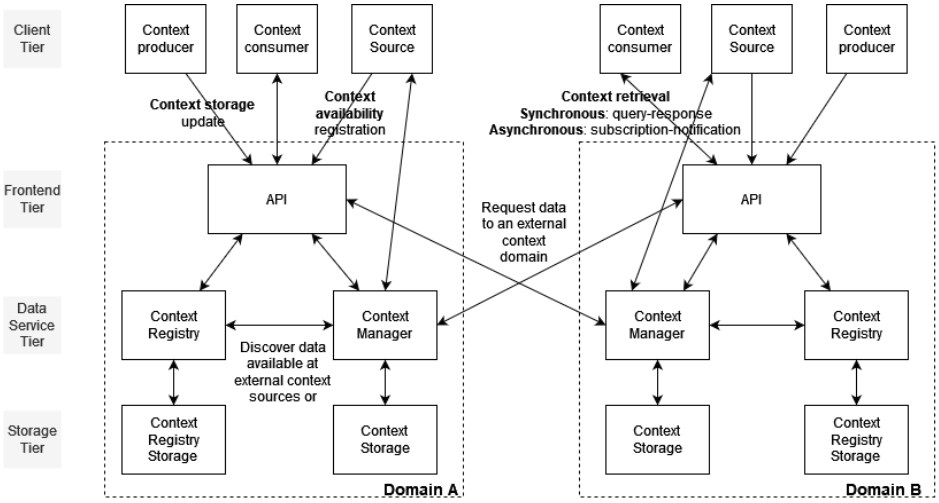

Figure C-2-1-1, Reference Implementation Model of the IDH Context Broker

In particular, we have the following components:

- **Context Producer**

  A context producer produces context (such as a sensor) and pushes it to the Context Broker. The storage managed by the involved Context Broker stores it.

- **Context Consumer**

  A context consumer requests context either synchronously or asynchronously. Context is requested with a context scope, such as data related to a thing, related to a type of a thing, related to a geographic scope, related to a temporal scope, or a combination of them. Synchronous requests follow the query-response pattern. The context consumer expects a single message response with the requested context. The requested context might be collected from any contacted domain, an external context source handled by the contacted domain, or an external domain. In the case of asynchronous requests, a continuous context stream is established towards the context consumer. As soon new matching context is available either within the contacted domain, an external context source, or an external domain, it is notified to the context consumer. The IOWN DCI might optimize the data traffic, especially for asynchronous communication, in order to avoid bottlenecks and reduce duplication of notifications. The subscription request might specify a time Interval value to configure a periodic notification of context. This configuration value should be taken into consideration for message exchange optimization.

- **Context Source**

  Context source is a source of context that exposes an API for context retrieval. This source can provision static context data, fresh sensor measurements, or the outcome of analytics processes.

- **API**

  API exposes a domain-specific language to access context data synchronously or asynchronously. API also exposes operations to register new context sources.

- **Context Manager**

  Context manager serves requests from context producers and context consumers. When a context producer sends a context update, the context manager stores it in the database and matches it against any established subscription. When a context consumer sends a context retrieval request, the context manager discovers context availability matching the request from the context registry and local index. The context might be available from zero to all of the following sources: domain context storage, locally managed context sources, and external domain. In a synchronous request, the context manager gathers context from all the involved sources and retrieves a single context message. In the case of asynchronous requests, the context manager establishes a subscription channel to any involved sources.

- **Context Registry**

  Context registry manages the list of available context sources and external domains with their respective set of contexts. The information contained in this component describes the topology of context sources and their available context. This information might be used for optimized traffic and intelligent context distribution.

- **Context Storage and Context Registry Storage**

  Basic service classes implement these components. A Message Broker, a Graph Store, and a Distributed RDB can be used simultaneously for a different part of context data. In particular, the Message Broker handles the continuous updates of context from context producers and towards context consumers. The Graph Store keeps the semantic representation of the context and the relationships between pieces of information. The Distributed RDB stores fundamental structured data such as spatial information and temporal information.

C.2.2. Converged DB

In today's typical cloud-based implementation model, each IDH service provides a limited set of capabilities only. For instance, the Relational DB can manage tabular format data but not other types of data. The Message Broker can accept the massive data ingestion and deliver it to the client application in almost the same order* the data is ingested

but does not support conditional read. Therefore, if the DX apparitions that IOWN Global Forum assumes are built on top of such limited IDH services, many IDH services need to be combined, and massive data must be transferred across many components.

* If several shards are used to pipeline the same stream, the order will not be guaranteed for inter-shard reads.

The Converged DB is a new concept that eliminates these inefficiencies and streamlines DX application development by offering multiple capabilities over different types and formats of data from a single box. Some example scenarios are 1) querying and joining the tabular and the semi-structured data such as parquet and/or JSON file together in one place, 2) running transaction processing and heavy analytical processing in parallel, and 3) supporting data extraction through the conditional queries and data analysis with its inherent compute resources while accepting a large amount of data ingestion.

To build such a Converged DB, more resources need to be dynamically coupled because workloads and data volume of the Converged DB will fluctuate significantly. On top of it, various software-level ingenuity needs to be implemented to optimize performance, such as data placement and the number of copies to be kept in the server cluster, indexing of the content, and the data organization in both the heap area and data persistence layer. The reference implementation model that reflects those points is described below:

- **Smart Client**

  Like the IDH Distributed RDB case, the clients will become smart to send the data access requests to the most appropriate Converged DB server where (most of) the request data exists in the cache.

- **Gateway**

  If the IDH Converged DB needs to accommodate millions of clients from various locations, it may be challenging to deliver data placement and data distribution awareness to all of the clients. In this case, there will be a gateway server tier that intermediates connection and data access requests from clients to the IDH Converged DB servers.

  The gateway may also include the security endpoint component if the IDH Converged DB is configured multi-tenant, as in the case of the IDH Distributed RDB.

- **Converged DB**

  In the IDH Converged DB, the DB Engine will become “smarter” to optimize the query optimization plan by considering the available resources at each server as well as the network latency and the bandwidth between servers. It will determine which DB server(s) to preprocess data, whether to transfer queries or data across servers when inter-server processing is required, where and how to join and combine the data, etc.

- **Storage**

  In the IDH Converged DB storage, a huge amount of data of multiple types is stored together. Various accelerators will be built in to speed up data processing in consideration of such a situation. For instance; i) an accelerate to encrypt and decrypt data with different encryption keys based on data content-aware policy, ii) an accelerate that encodes and decodes in an optimal way according to the content of the data, and iii) an accelerate to manage summary index that describes the maximum and minimum values of each column for each fixed number of records, will be used.

- **Smart Client - DB Server Connection**
- **Inter-DB Server Connection**
- **DB Server - Storage Connection**

Figure C-2-2-1 shows images of the IDH Converged DB reference implementation model with such ingenuity described above.

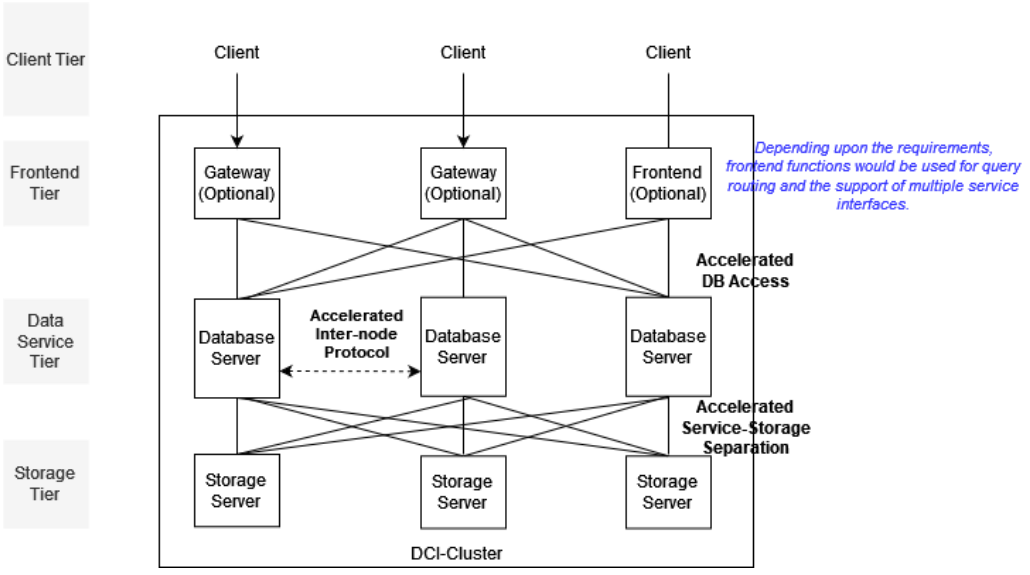

Figure C-2-2-1, Reference Implementation Model of the IOWN IDH Converged DB

C.2.3. Virtual Data Lake (Federated Storage)

In the future digital era, more data generated from different locations will be managed in a distributed manner, such as in the edge clouds. Besides all that, the Object Storage and the Distributed File Storage will remain a core component to store big data as it provides the cheapest data storage mechanism. This means some orchestration feature needs to be built so that distributed data can be well managed, which is called “IDH Virtual Data Lake”.

Figure C-2-3-1 shows such an IDH Virtual Data Lake that integrates multiple storage services. Each storage service may belong to a different organization, but each must be configured to be trusted and federated with one another when configured as part of the Virtual Data Lake.

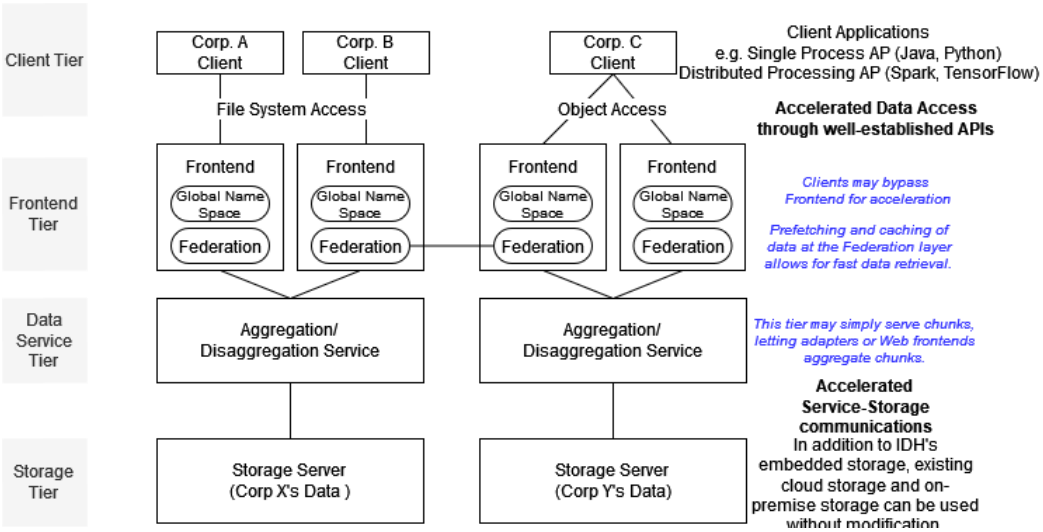

Figure C-2-3-1, Reference Implementation Model of the IDH Virtual Data Lake

To realize a fast and quality Virtual Data Lake, several ingenuities will be added on top of the Object Storage and the Distributed File Storage. Below is a list of the components and network connections that make up the IDH Virtual Data Lake and explanations of ingenuity implemented, if any.

- **Client**

  To realize the data transfer between multiple data owners of IDH, those tenants should be able to get data managed by other tenants through a general API or I/F regardless of location and data storing method.

- **Frontend**
  - Global Name Space and Unique Object or Directory ID

    By designating the globally integrated/unified namespace and object or directory ID, the clients can access any objects or directories without being aware of their location and owner. (E.g., a client application can get a file object as if it gets the file from a directory of the local storage).

  - Federation

    Unlike the basic Object Storage which stores all object data in a single place, the IDH Virtual Data Lake can manage data stored in a distributed environment. For this purpose, the IDH Virtual Data Lake federates actual data access requests to the remote IDH Object Storage and/or to the remote Distributed File Storage with the support of the global name space and the unique object/directory ID. Using this new approach, a client application can access the necessary data easily without knowing actual deployment of the IDH services. By doing so, the IDH solution avoids creating the enormous amount of data copies that occur in today's implementation model. The IDH Virtual Data Lake is configured to ensure high-speed and efficient on-demand data transfers by leveraging the IOWN DCI architecture and the IOWN OpenAPN architecture. Cache sharing function and pre-fetch function in accordance with the access pattern and frequency will also be provided.

  - Basic authentication and authorization functions are provided to allow data access only for authorized clients. This model is equipped with an access control function unique to a data distribution platform that takes into account data provenance and lineage (e.g., data from company X and data from company Y is jointly processed by company A). Furthermore, as an optimization technique for multi-tenant premises, it is equipped with data acquisition speedup by sharing cache between tenants, data acquisition/deployment functions considering the locality of each component, and so on. To realize such a function, frequent data transfer is required within the frontend tier (between Federation components), but the effect can be minimized by improving the efficiency of data transfer by the IOWN DCI.

- **Aggregation / Dis-aggregation Server**
- **Storage Tier**
- **Client - Web Frontend Connection**
- **Web Frontend - Aggregation / Dis-aggregation Server - Storage Connection**

C.2.4. Virtual Data Lake House

As there will be many data services to be used because of the diversity of business requirements, data from various data sources need to be analyzed and used together. Therefore, a special data service called "IDH Virtual Data Lake House" is required to support such requirements.

Figure C-2-4-1 shows a reference implementation model of the IDH Virtual Data Lake House that virtually aggregates and manages data from various data sources belonging to multiple data owners.

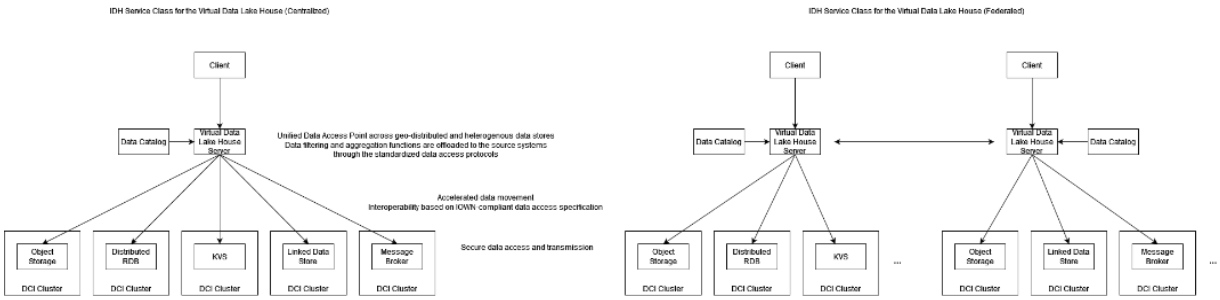

Figure C-2-4-1, Reference Implementation Model of the IDH Virtual Data Lake House

To enable such an IDH Virtual Data Lake House concept, several ingenuities will be introduced on top of other IDH services. Below is a list of the components and network connections that make up the IOWN Virtual Data Lake House and explanations of ingenuity implemented, if any.

- **Client**
- **Data Catalog**

  To provide virtualized data access across multiple data services, data services and statuses of data stored in each data service must be known. The Data Catalog provides such information to the Virtual Data Lake House servers.

- **Virtual Data Lake House Server**

  This component will provide a single point to access data for its client on top of various IDH services. That means, if the client application publishes an SQL query to the Virtual Data Lake House, then the Virtual Data Lake House disassembles the query by considering other IDH services to be contacted and forwards these disassembled queries to them. Then, each IDH service that has received such a disassembled query will preprocess data as requested and reply only to the Virtual Data Lake House. Through such a mechanism, global data access is streamlined.

  To provide unified access across multiple IDH services, wrapper components that manage their interaction are added. In addition, inter-server parallel processing is performed on multiple Virtual Data Lake House servers to combine data from multiple IDH services and obtain the final result.

  In addition, it is possible to federate multiple Virtual Data Lake Houses. To realize a better society, it is necessary to combine and utilize various data from multiple companies safely. The Virtual Data Lake House will be a foundation for such a better future.

- **Virtual Data Lake House Server - Other IDH Service Connection**

  To accelerate data access, connections between the Virtual Data Lake House server and other IDH services will be accelerated by the IOWN OpenAPN and the IONW DCI technologies.

Appendix D: Example Use Cases

D.1. Live 4D Map for the CPS Area Management Use Cases

D.1.1. Use Case Overview

To realize the future area management services described in the IOWN Global Forum use case reports, it will be necessary to collect various data from the field sensors, analyze it to understand the real-world situation it is measuring, detect the events of concern, and record it so that all important information is preserved.

In particular, it would be almost impossible to realize an application that comprehends real-world situations and detects events of interest without a well-designed data management infrastructure that supports processing massive data for the purposes of analytics. From such a point of view, it is necessary to develop a so-called Live 4D Map data management infrastructure. The positioning of the Live 4D Map in the system is shown in figure D-1-1-1.

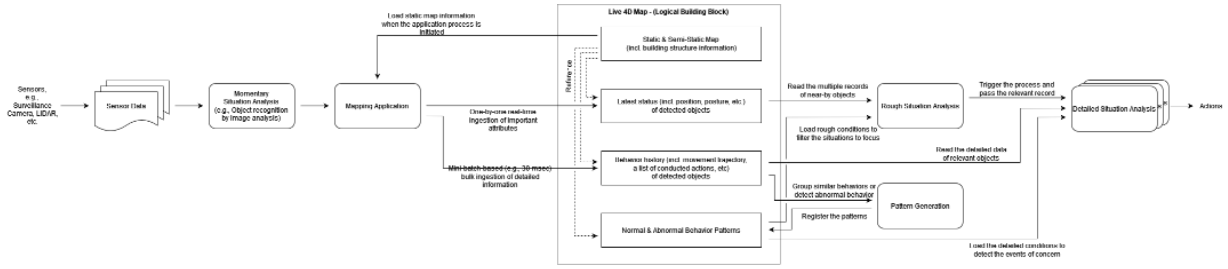

Figure D-1-1-1, Positioning of the Live 4D Map within the CPS Area Management Security system

As described in figure D-1-1-1 and figure D-1-1-2, four types of data need to be managed in the Live 4D Map:

- Static and Semi-Static Map

  This data represents underlying structural information of the monitored area. For example, it includes the interior walls, entrances, and doors of a building, and fixtures such as desks and chairs there. It should be noted that fixtures such as desks and chairs are sometimes moved, but not often. As such, they are referred to here as semi-static map information.

- Recent Status of Detected Objects

  This data represents the summary information of detected objects such as people, cars, and dangerous objects. This data will include pseudonymous tracking ID, positions, postures, owner-and-belongings relationships, risk category, etc. which may be added up to several KB per record in total. When building this layer, the following requirements must be met:

- Behavior History of Detected Objects

  Since the detected object moves and does different actions over time, such history needs to be tracked. To support detailed analysis, this data preserve quite detailed information, such as positions, postures, actions, etc. It should be noted that the action information is extracted by running a bit of time-series analysis. Therefore, the mapping application in the previous figure needs to preserve the data for a while even after the data is ingested into the Live 4D Map.

- Normal & Abnormal Behavior Patterns

  This data represents the expected normal behavior patterns of objects such as people, cars, etc., and abnormal behavior patterns, such as entering the restricted zone, etc. Having this kind of pattern information enables efficient situation analysis. Such patterns may not be apparent when opening the building. In that case, it is registered by analyzing the behavior history data described above.

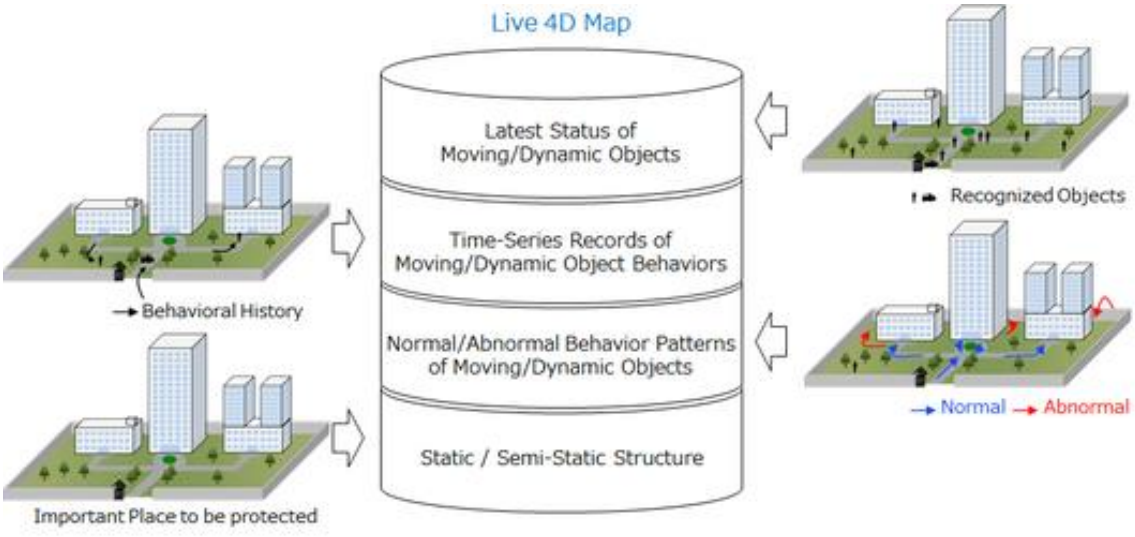

Figure D-1-1-2, Live 4D Data Structure

On top of such a data structure, there will be the following applications to build the area management security service.

- Momentary Situation Analysis

  This application extracts momentary situation insights by analyzing the sensor data. For instance, this analysis could include image analysis to detect objects from the image collected.

- Mapping Application

  This application determines the position of the detected objects within the registered map. To determine the position of the detected objects, this application may leverage LiDAR sensor data, or perform visual analysis by referencing static map information to calculate the relative position of the detected objects to the static structure elements.

- Pattern Analysis

  This application analyzes the accumulated behavior history information. By understanding the standard human behavior in the building, it becomes possible to quickly extract abnormal behavior patterns. In order to efficiently identify normal patterns, the similarity of behavioral histories is evaluated, similar behavior records are grouped together, and grouped records are statistically processed.

- Rough Situation Analysis

  This application extracts the events that need more attention by comparing the latest status with summarized normal and abnormal patterns. By just comparing two data at the summary level, this application can determine such a situation rapidly and efficiently. Once such events are determined, this application triggers the subsequent process, one or many detailed situation analyses.

- Detailed Situation Analysis

  This application analyzes the situation deeply by checking the detailed behavior history records and comparing them with normal and abnormal behavior patterns. For anomaly detection, the dissimilarity between the behavior history and the normal behavior patterns, or the similarity between the behavior history and the abnormal behavior patterns is calculated.

D.1.2. Data Management Requirements

Depending upon the use case scenario, the throughput and response time requirements vary. However, it would be reasonable to assume the following metrics will be necessary to satisfy advanced IOWN use cases, as well as the associated IDH services needed to satisfy these performance goals.

- Latest Status Tier:
  - Accommodate tens of thousands of records that can be several KB in size.
  - Accept in excess of millions of updates per second at a latency of milliseconds or less to update the statuses of moving objects to be monitored, e.g., positions, actions, relationships with other objects, etc.
  - Respond very quickly to the query that retrieves the information of target objects at a latency of a few milliseconds or less. This provides sufficient computing time that can be reserved for the rough and detailed analyses to meet the end-to-end response time objective of less than 1 second or ideally 100 milliseconds.
- Behavior History Tier:
  - Accommodate more than billions or trillions of data records, each of which has several KB or more.
  - Accept more than millions of inserts per second to record real-world events, object movement, actions, etc.
  - Respond to the query that retrieves historical records of objects of concern at a latency of tens of milliseconds or less, to support analysis of historical movements/behaviors of the objects.
- Behavior Pattern Tier
  - Manage as many patterns as needed.
  - Manage interactions and relationships between two or many objects.
  - Compare each monitored object behavior with registered patterns, at a latency of 10 milliseconds or less.
- Static and Semi-Static Map Tier
  - Manage geometry objects precisely, e.g., with cm-level accuracy
  - Support both the geospatial query and the update transactions quickly, at 10 milliseconds or less latency.

    Note: The semi-static structure includes desks, chairs, etc., which could be moved occasionally

D.1.3. IDH Solution

It is apparent to say that to build the Live 4D Map, multiple types of data need to be jointly managed and analyzed with very low latency and in a very scalable manner. To meet these challenging requirements, various IDH services that are built on top of the OpenAPN and the IOWN DCI will be used and combined. Figure D-1-3-1 shows a possible high-level design of the Live 4D Map.

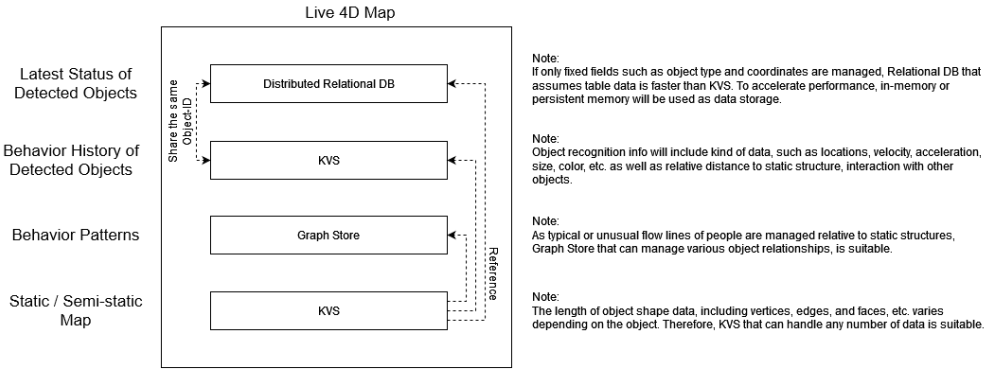

Figure D-1-3-1, IDH Service Mapping to build the Live 4D Map

The technical points of each IDH service used for the Live 4D Map are described below:

- **Distributed RDB**
  - The Distributed RDB can only handle fixed columns in tabular format but provide the best performance for data updates and queries because it knows the data structure in advance. Therefore, it is used to manage the data that needs to be updated and is common across various objects such as coordinate information of detected objects.
  - When the data record is updated within the Distributed RDB, a pointer to the corresponding time-series records will be managed in the record. Also, reference information to the static/semi-static structure information will be added to the record.
  - The Distributed RDB, if properly designed with the IOWN Global Forum technology, can support hundreds or tens of millions of TPS of data update and query processing, and such performance is essential for managing the dynamic data used to track a massive number of objects in real time.
  - The Distributed RDB needs to be connected to the relevant applications through the IOWN OpenAPN so that data processing can be executed in a real-time manner.
  - The Distributed RDB needs to be connected with the KVS and the Graph Store through the IOWN OpenAPN or use the Converged DB, so that data can be referenced each other and data validity can be checked in a real-time manner.
- **KVS**
  - The KVS can manage various events flexibly in a scalable manner. Since the output from AI inference processing can include various attributes, data flexibility of the KVS is required. Also, data flexibility is required to manage the static and semi-static structure information within the Live 4D Map, because structural elements have different shapes and attributes. Therefore, it is used to manage the time-series records of detected objects and also the static and semi-static structure information within the Live 4D Map.
  - To accelerate the query, adequate indexes will be created for each record. For instance, to well-support the position-based query, spatial indexes will be created.
  - As massive data ingestion and management are required, the KVS is designed to scale linearly without sacrificing performance by utilizing the IOWN Global Forum technology.
  - The KVS needs to be connected to the relevant applications through the IOWN OpenAPN so that data processing can be executed in a real-time manner.
  - The KVS needs to be connected with another KVS, the Graph Store, and the Distributed RDB through the IOWN OpenAPN or use the Converged DB so that multiple data sources can reference each other and data validity can be checked in a real-time manner.
- **Graph Store**
  - The Graph Store can manage relationships between different data efficiently. To represent normal and abnormal behavior patterns, it is necessary to manage the relationships of various data in chronological order. For example, the pattern record may look like; 1) picking up the goods from the shelf, 2) putting them in the bag, and 3) leaving the store without paying for them. It would be appropriate to manage such pattern information in a Graph Store.
  - The Graph Store needs to be connected to the relevant applications through the IOWN OpenAPN so that data processing can be executed in a real-time manner.
  - The Graph Store needs to be connected with the KVS and the Distributed RDB through the IOWN OpenAPN or use the Converged DB so that data can be referenced each other and data validity can be checked in a real-time manner.

D.2. Mobility Digital Twin for the CPS Mobility Management Use Cases

D.2.1. Use Case Overview

Future mobility services require a new infrastructure that monitors and predicts traffic situations in the area in real-time. For example, information such as information on dangerous objects on the road is used to reduce the speed of vehicles that approach the point to avoid danger, and current and future traffic congestion information is used for route selection for user vehicles and wide-area traffic control to reduce energy consumption and increase passengers' benefit. Such information will be collected from running vehicles and roadside sensors and reflected on the mobility digital twin platform.

However, such data management is a heavy burden. With millions of intersections and road segments to monitor and manage in a city, the data management platform shall be able to grasp the situation of dynamic objects around intersections every second or less, and to monitor and predict the traffic congestion of each road segment continuously, for example, every 15 seconds. This means it must be able to respond to data access requests from millions of vehicles and user devices while updating large amounts of data at any time.

The IDH services are used to meet such stringent demands. The IDH services record history of traffic congestion and on-road events such as accidents, manage current traffic congestion and on-road events and predict future traffic congestion on top of static road structure information. Figure D-2-1-1 below shows a conceptual design of a mobility digital twin developed by applying the IDH service.

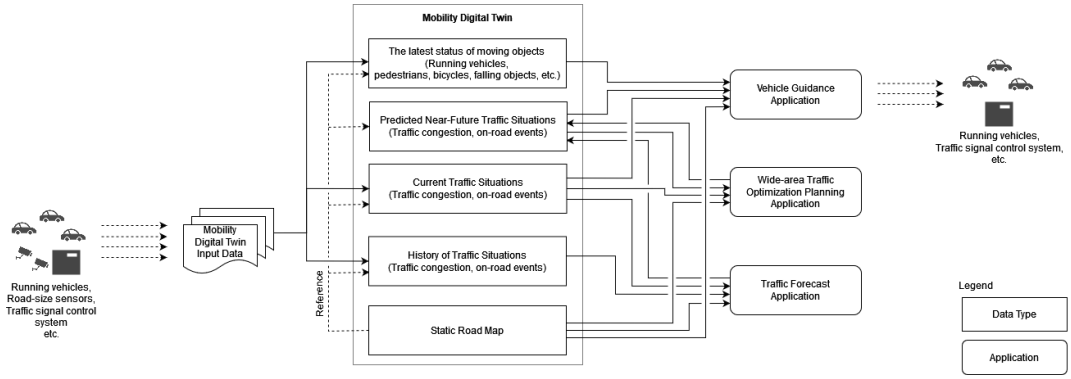

Figure D-2-1-1, Positioning of the Mobility Digital Twin within the CPS Mobility Management system

As described in the above figure, five types of data will be managed in the Mobility Digital Twin platform:

- Static Road Map

  This data represents the static structure of the road, such as road shape, lane structure, road painting position, slope, road surface condition, and structures, such as safety fences around the road. These data do not change frequently, but since the total length of roads in one country is hundreds or tens of millions of kilometers, the total amount of data is tens or hundreds of terabytes. Also, on top of this physical structural data, road segments and their linkage information will also be generated as administrative data, and the degree of traffic congestion is managed per road segment. The road segments and their network data are, to some extent, artificially generated, but they are also generated automatically based on intersections and/or information about location points where the different traffic regulations start becoming effective.

- History of Traffic Situations

  This data represents statistics about traffic congestion and on-road events such as accidents. In order to record such information for each location, the administrative data described above will be used. In other words, it records and manages the traffic volume and the number of accidents by time period for each road

segment. To perform highly accurate traffic optimization, statistics data will be created for every 100-meter road segment every 5 minutes. In that sense, the granularity of historical traffic situation data is less than the current traffic situation data.

- Current Traffic Situations

  This data represents current traffic congestion degrees as well as ongoing on-road events. To manage the location of these data, the same administrative data is used, however, more detailed information such as the start and end point of each traffic jam, and the exact location of each on-road event. It should be noted that this information will need to be updated frequently, such as every 15 seconds, and sometimes in an ad-hoc manner so that wide-area traffic optimization is made in an effective manner to reduce energy consumption by running cars and the safety maneuver is applied in a timely manner.

- Predicted Near-Future Traffic Situations

  This data represents the forecasted traffic congestion degrees in the very near future, such as the next 4 hours. The forecast is not only determined by considering the historical and/or seasonal trends of traffic congestion, but also by analyzing the impact of current ongoing on-road events. For instance, if traffic congestion occurs because of snow, then the amount of accumulated snow and the weather forecast is considered. Such forecasted information is also updated frequently so that the objectives of the mobility digital twin are achieved.

- The Latest Status of Moving Objects

  This data represents the status of the moving object, such as the position of running vehicles and pedestrians. This information is very important for the safety maneuver, because, for example, if an accident happens, then the notification should be sent to the vehicles that approach that point, so the position of each vehicle that subscribes to the safety maneuver service should be managed. This information needs also to be updated very frequently, such as every second. Otherwise, it will not be able to avoid a collision with a pedestrian who inadvertently jumps into the road.

In addition, there are the following applications (application functions if these are implemented as a monolithic application) that interact with the mobility digital twin platform.

- Traffic forecast application

  This application creates a forecast of the traffic situations for each of the road segments for the next few hours every several tens of seconds. To make a precise forecast, current traffic situations, historical traffic situations, ongoing on-road events, and other supplemental information such as weather forecasts are considered.

- Wide-area traffic optimization planning application

  This application determines the optimized route plan if a running vehicle requests the application to find it, and/or determines the optimized distribution of traffic across various possible routes for the vehicle fleet of concern to minimize the vehicles' energy consumption as a whole.

- Vehicle guidance application

  This application provides the safety maneuver as well as other valuable information to the vehicles as required. For instance, if an accident happens, then urgent safety notifications are sent to the vehicles that will arrive at the point within several tens of seconds, and if traffic congestion happens, then recommendation messages of the route change are sent to the vehicles that plan to pass through the point where the congestion happens so that they can avoid being involved by the traffic jam.

D.2.2. Data Management Requirements

Assuming that there are one hundred of thousand intersections, one million road segments, ten million running vehicles, and 1 million concurrent moving objects such as pedestrians, bicycles, etc. to be monitored, the following requirement must be considered for each component relevant to build the mobility digital twin platform.

- The Latest Status of Vehicles and Moving Objects
  - Accommodate more than millions of moving object records, as there will be more than a few running vehicles in a large city.
  - Support more than millions of updates per second, as millions of the moving object statuses need to be updated frequently.
  - Run the query with sub-second latency while accepting massive updates of data. This is a critical requirement to achieve the safety maneuver.
- Predicted Near-Future Traffic Situations
  - Accommodate more than hundreds of millions of forecast records, because there will be more than millions of road segments, and forecasts will be made for the next few hours with a few minute time buckets.
  - Support regular updates of all data, as the forecast data needs to be refreshed periodically to achieve the objectives.
  - Support more than thousands of queries per second with sub-second latency, because each running vehicle may query the forecasted traffic situation every 10 minutes or so, and the optimized traffic route needs to be returned in less than a few seconds.
- History of Traffic Situations
  - Preserve more than trillions of records, as the traffic situation history needs to be stored for more than years, and there are millions of road segments to be distinguished and the time-bucket size will be several tens of seconds.
  - Support queries that may extract thousands of records by specifying the primary key, i.e., road segment ID, so that the traffic situation forecast can be updated in a timely manner.
- Static Road Map
  - Store more than hundreds of millions of records, as there will be more than hundreds of different entities created for each of millions of road segments, which can be more than TB in total if the size of each record is about 10 KB.

D.2.3. IDH Solution

To build the mobility digital twin infrastructure, several IDH services are combined, as there are multiple types of data to be managed as described above. Figure D-2-3-1 shows a possible high-level design of the mobility digital twin infrastructure based on IDH services.

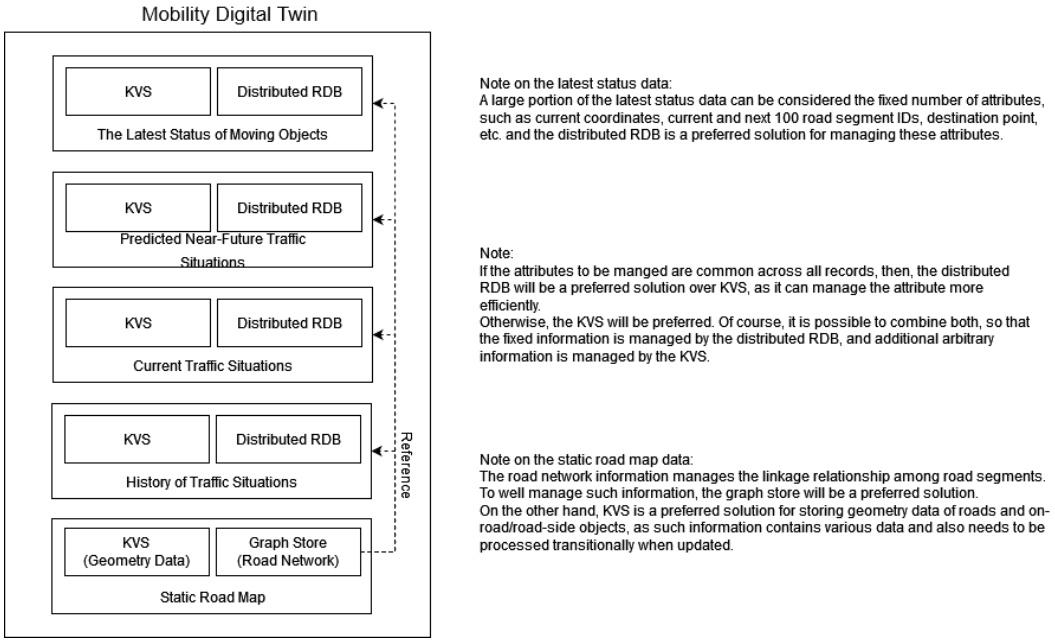

Figure D-2-3-1, IDH Service Mapping to build the Mobility Digital Twin

The technical points of each IDH service used for the mobility digital twin are described below:

- Distributed RDB
  - The Distributed RDB will be used to manage 1) the latest status of moving objects, 2) predicted near-future traffic situation, 3) the current traffic situation, and 4) the history of the traffic situation because these information records will only contain a fixed number of attributes.
  - An in-memory-based Distributed RDB will be used for the predicted near-future traffic situation and the current traffic situation because massive data updates need to be supported in an efficient manner. In the future, a remote memory pool connected to multiple compute servers through the IOWN OpenAPN will be used for better performance, scalability, and efficiency.
  - To run location-based queries very fast, indexes for the road segment ID and coordination attributes are created and used.
- Graph Store
  - The Graph Store will be used to manage the road network information because it can navigate through the network structure very quickly and efficiently.
  - To support various geo-queries, the spatial index will be created on top of graph data.
  - The graph data model used will be hierarchical because the required granularity of the road structure information differs depending on the needs.
  - The Graph Store is connected to the Distributed RDB through the IOWN OpenAPN so that the traffic situation monitoring and analysis can be done in a real-time manner.
- KVS
  - The KVS will be used to store the shape and arbitrary attribute data about roads.
  - Such data is used to determine the safety maneuver because otherwise how to operate the steering wheel to avoid danger cannot be decided.
  - It should be noted that relevant data stored in the Graph Store and the KVS needs to be extracted simultaneously so that rapid decision-making is supported.

D.3. Manufacturing Digital Twin for the CPS Industry Management Use Cases

D.3.1. Use Case Overview

The manufacturing digital twin model was first introduced by Michael Grieves at a Society of Manufacturing Engineers conference in 2002, which was designed to support product lifecycle management. Since then, the digital twin has been continuously discussed in the manufacturing and engineering industry (as well as NASA, the US DOD, etc.), and a framework has subsequently been developed. The mobility digital twin described above was also inspired by this initial manufacturing digital twin model and has also been discussed by the connected car industry.

To support the product lifecycle, the manufacturing digital twin is divided into four data domains:

- Digital Twin Instance (DTI)
- Digital Twin Aggregation (DTA)
- Digital Twin Prototype (DTP)
- Factory Structure/Layer

Figure D-3-1-1 shows an adoption image of such a manufacturing digital twin.

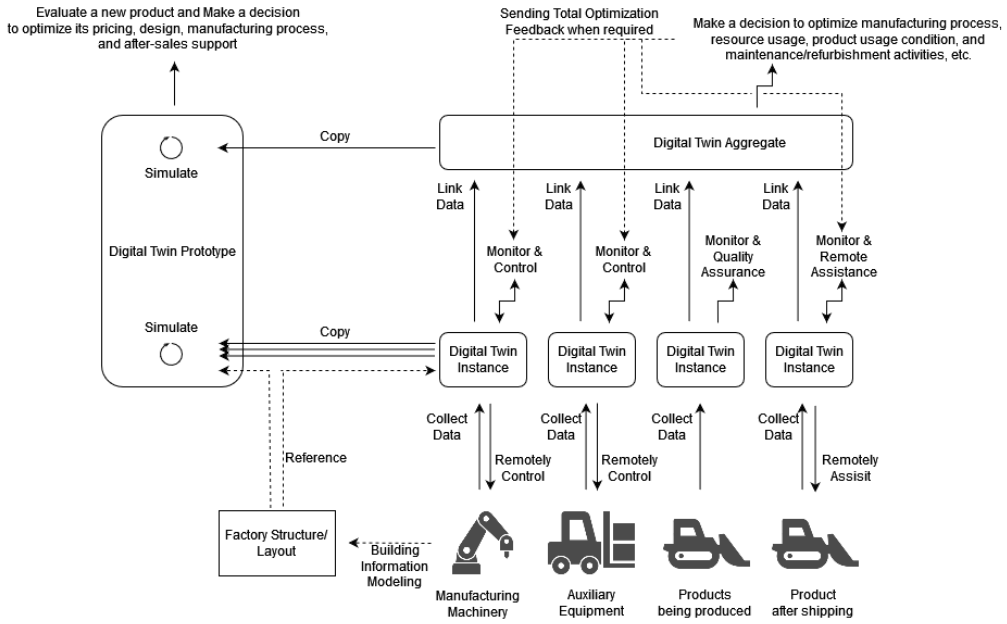

Figure D-3-1-1, Positioning of the Manufacturing Digital Twin within the CPS Industry Management system

As described in the above figure, four different domains of data will be managed in the Manufacturing Digital Twin platform:

- Digital Twin Instance (DTI)

  DTI data is used to represent the status of each physical object, such as work-in-process materials, machinery and equipment in the factory, shipped products, etc. Such information is not only used for monitoring but also for remote controls and assistance. To reflect the physical world situation, various sensor data are collected.

- Digital Twin Aggregate (DTA)

DTA data is used to aggregate multiple DTI data so that the entire target physical system can be monitored and analyzed.

- Digital Twin Prototype (DTP)

  DTP data is created and used to support the planning, design, and evaluation of new products before their production starts. To evaluate the expected production efficiency, durability, and usage value of a new product before its actual production, the expected situation will be simulated in the DTP. By running the simulation, and comparing multiple simulated plan versions, the product design, the manufacturing process design, and the usage scenario can be optimized.

- Factory Structure/Layout

  Factory Structure/Layout data is created to model the structure of the target factory. DTI and DTP data are managed with a reference to this data. The factory structure/layout data is created through a so-called building information modeling (BIM) process.

D.3.2. Data Management Requirements

Assuming that 1) manufacturing digital twin infrastructure is built for managing production processes, 2) there are 10,000 machines and robots in a target factory, 3) each robot or machine is remotely controlled with less than a 10-millisecond latency, 4) factory-wide control information, such as knowledge of nearby robots that may collide each other, are updated every second, and 5) there are 1,000,000 concurrent work-in-process (WIP) materials that need to be monitored every second, the following requirements must be considered for each data domain described in the previous sub-section.

- Digital Twin Instance (DTI)
  - Manage and record the status of 10,000 machines and/or robots, and 1,000,000 WIP materials
  - Update the status of 10,000 machines and robots in far less than 10 milliseconds
  - Update the status of 1,000,000 WIP materials every second with sub-second latency
  - Respond to the query that retrieves the status information of each machine or robot in far less than 10 milliseconds
  - Respond to the query that retrieves relevant status information of multiple machines, robots, and WIP materials with sub-second latency
  - Record the DTI data change history for a longer period of time, such as 10 years, to support the big data analysis to improve production efficiency
- Digital Twin Aggregate (DTA)
  - Update the relationship between DTI by linking and unblinking the pointers dynamically with sub-second latency
  - Respond to the query that analyzes the DTI relationship information with sub-second latency
  - Record the DTA data change history for a longer period of time, such as 10 years, to support the big data analysis to improve production efficiency
- Digital Twin Prototype (DTP)
  - Load relevant DTI and DTA change history data rapidly, such as 30 minutes
  - Adjust/modify the loaded data according to the expected product and production process change, so that various scenarios can be simulated and evaluated
  - Support analysis on production efficiency, production lead time, production cost, etc.
- Factory Structure/Layer
  - Represent the structure and the shape of the factory building

- Represent the location and shape of each machine and robot in a coordinated system defined for the factory building

D.3.3. IDH Solution

To build the manufacturing digital twin infrastructure, the above data domains need to be managed and integrated in an effective and efficient manner. For this purpose, the IDH services are adopted as described in figure D-3-3-1.

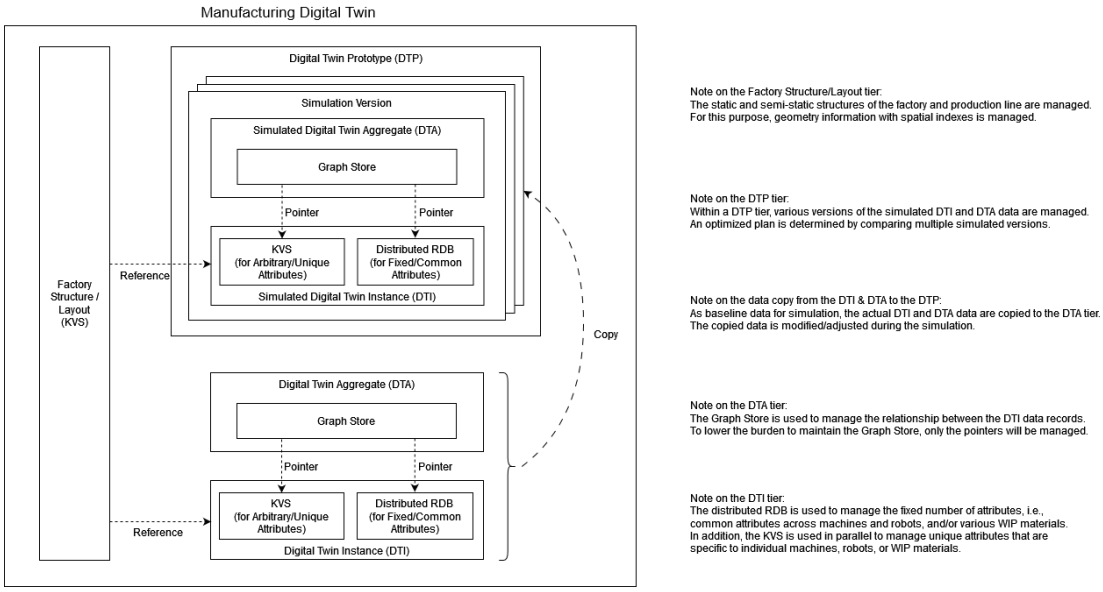

Figure D-3-3-1, IDH Service Mapping to build the Manufacturing Digital Twin

The technical points of each IDH service used for the manufacturing digital twin are described below:

- Distributed RDB
  - The Distributed RDB will be used to manage the common attributes across machines and robots or different types of WIP materials. Different tables will be created in the Distributed RDB for the machine and robot data, and the WIP material data.
  - The Distributed RDB needs to be configured to support massive data updates as well as primary-key-based queries so that each DTI can be monitored and remotely controlled in a real-time manner. For this purpose, an in-memory-based Distributed RDB may be used.
- KVS
  - The KVS will be used to manage an arbitrary number of attributes of machines, robots, and WIP materials. Such attributes are considered unique to an individual machine, robot, WIP material type, and WIP material. As the data structure does not need to be decided in advance, a new machine can be immediately integrated into the system when procured.
  - The KVS will be also used to manage the factory structure/layout information. However, an instance of the KVS will be different, as these data management requirements are different. To effectively manage the factory structure/layout, spatial indexes will be created within the KVS.
- Graph Store
  - To aggregate different data, it would not be reasonable to populate all the data from multiple DTIs. Instead, it would be appropriate to manage only a pointer to each DTI data in the DTA tier and develop data access technology that can extract both data quickly and efficiently.

- In that sense, the Graph Store will manage pointers to the data records of 10,000 machines/robots and 1,000,000 WIP materials stored in the DTI tier.
- The Graph Store is connected to the Distributed RDB and the KVS both of which store DTI and DTA data through the IOWN OpenAPN so that relevant data can be retrieved with short latency.

D.4. Network Digital Twin for the CPS Network Infrastructure Management Use Cases

D.4.1. Use Case Overview

As explained in the Open All-Photonic Network Functional Architecture document [IOWN GF APN FA], the IOWN OpenAPN will provide high-performance and quality network service, which is a critical element in realizing future digital experience services. However, the optical network requires a wavelength-based path to be established between two selected endpoints all the time when the networking is used. Therefore, it is necessary to predetermine how wavelength-based paths are established in the physical network topology.

Also, in terms of an ideal form of network-as-a-service, it is important to be able to accommodate a large number of different users within the same physical network topology. In addition, it will be valuable to vary the service quality, such as guaranteed bandwidth and latency of each provisioned network by time depending on the user’s need. With such characteristics, it becomes possible to provide high-quality and high-performance network services at a lower cost and with larger flexibility.

Achieving such goals requires a data-intensive mechanism. This is because it is required to 1) collect telemetry information continuously and frequently from a bunch of network devices, 2) map collected information on top of the physical topology model to understand the network service condition as a whole, and 3) make a dynamic decision to exclude malfunctioning devices and elements from the network, and 4) provision, reclaim, and change wavelength-based paths in an optimized manner according to users' demands.

The infrastructure that supports such data-intensive processing is called the network digital twin infrastructure. Within the network digital twin infrastructure, the network structure and the network operation status, such as provisioned paths are managed. Figure D-4-1-1 shows a very simple example of the IOWN OpenAPN structure and the status of provisioned wavelength-based network paths.

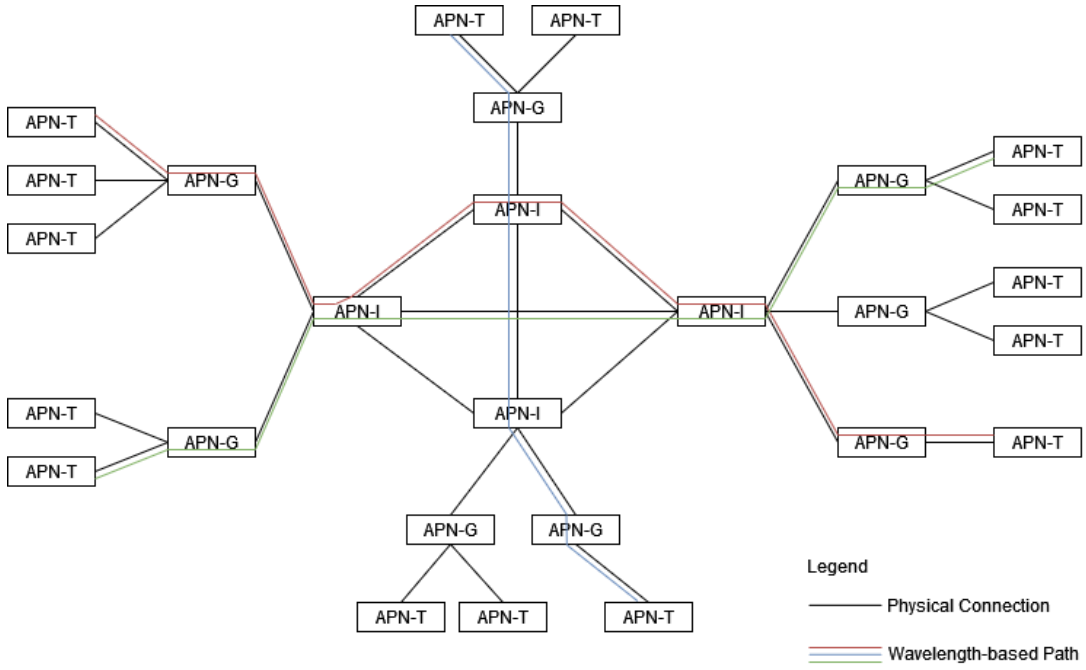

Figure D-4-1-1, An example of the IOWN OpenAPN system and provisioned wavelength-based paths

To effectively manage the IOWN OpenAPN system described above, the following data need to be managed at a high level:

- Underlying Network Topology
- Wavelength-based Provisioned Paths
- Telemetry data to
  - represent the latest status of the network devices and the stability and actual performance of each provisioned path
  - log the network operations status and issues

Figure D-4-1-2 shows an image of the content, flow, and positioning of each data within the network digital twin.

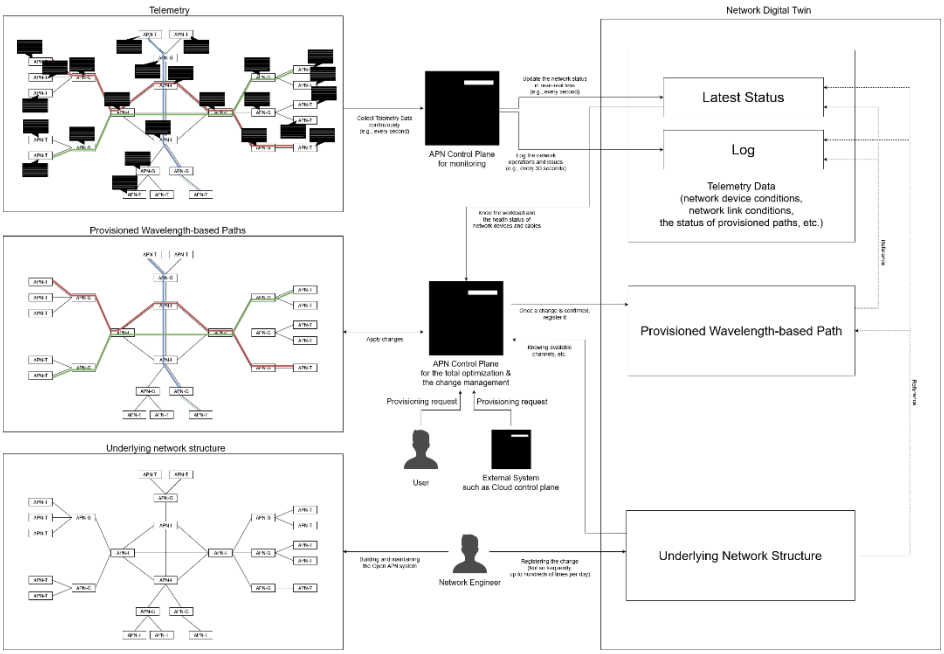

Figure D-4-1-2, Network Digital Twin Data Structure and Data Flow

D.4.2. Data Management Requirements

Assuming that a target IOWN OpenAPN system contains 1) 100,000 network devices, 2) 2,00,000 network optical fibers, and 3) 200,000 concurrent provisioned wavelength-based paths to be supported, and 4) telemetry data is collected, monitored, and analyzed every second, then the following requirement must be considered for each data domain described in the previous sub-section.

- Underlying Network Model
  - Accommodate active records of 100,000 network devices and 2,000,000 links.
  - Record the historical changes of the network devices and links by classifying the old record as obsolete. The historical data may be preserved for years.
- Wavelength-based Provisioned Paths
  - Accommodate active records to manage 2,000,000 wavelength-based paths.
  - Find the best path that satisfies the provisioning request in less than a minute.
  - Optimize the overall provisioning of paths so that energy consumption is minimized while assuring all customer QoS objectives are met.
- Telemetry Data - Latest status
  - Update the status data of all network devices and links every second.
  - Alert the abnormal situation by continuously running analytical queries.
- Telemetry Data - Log
  - Record all the detailed logs.
  - Support analysis to find the root cause if any problem occurs

D.4.3. IDH Solution

To build the network digital twin infrastructure, a multi-layered data organization will need to be established by the network operator of concern, and the above different four types of data need to be managed and integrated in an effective and efficient manner Therefore, a special IDH solution needs to be developed for this purpose so that the multi-layered data organization can fulfill its responsibilities easily. A high-level configuration example of such an IDH solution is shown in figure D-4-3-1:

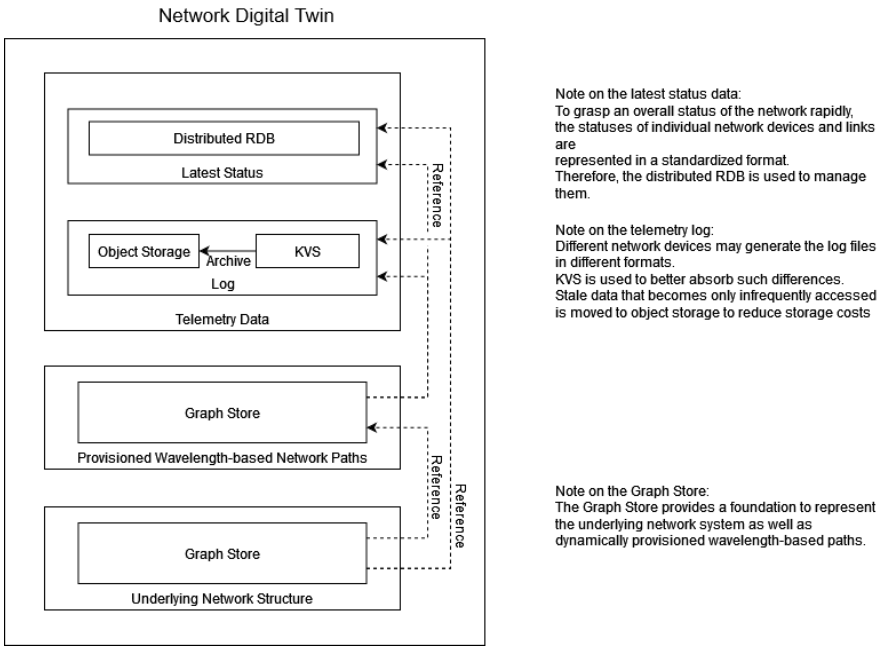

Figure D-4-3-1, IDH Service Mapping to build the Manufacturing Digital Twin

The technical points of each IDH service used for the network digital twin are described below:

- Distributed RDB
  - The status of the network devices and network links are represented in a standardized format within the Distributed RDB.
  - Different tables will be defined for different target entities such as APN-T, APN-G, APN-I, cables, and provisioned paths.
  - In-memory-based implementation model may be used as there are millions of records that need to be updated every second.
- KVS
  - The KVS is used to store the various log files which might have different formats in a flexible and scalable manner.
  - Stale data that becomes only infrequently accessed is moved to the Object Storage to reduce storage costs.
- Object Storage
  - The Object Storage is used to store stale data in a cost-effective manner.
  - By using the query acceleration mechanism of the IDH Object Storage, it is possible to analyze data in the Object Storage at a certain speed.
- Graph Store

- The Graph Store is used to represent the underlying network system as well as dynamically provisioned wavelength-based paths.
- The underlying network system data and as dynamically provisioned wavelength-based path data are managed as a different layer of the graph data but can be jointly processed rapidly.
- The Graph Store is connected with the Distributed RDB through the IOWN OpenAPN so that an optimal plan to provision the network path(s) can be found quickly.

D.5. Smart Grid Digital Twin for the CPS Smart Grid Management Use Cases

D.5.1. Use Case Overview

Renewable energy is generated in a fairly decentralized manner, and the amount of power generated by each power generation facility is unstable. Therefore, if renewable energy is used as the main power source, stable operation of the entire power grid system will be extremely difficult to maintained as described in the IOWN Global Forum CPS Use Case document [IOWN GF CPS UC].

To overcome these limitations, the smart grid digital twin is expected to be a solution that the entire power grid system to be operated in a stable manner with renewable energy resources.

In this context, the smart grid digital twin is expected to provide a foundation to precisely 1) predict the amount of power generated by each power generation facility, 2) predict when and how much power will be consumed by power consumers involved, 3) manage the status and the planned use of distributed energy storage resources such as electric vehicles, 4) determine the optimal supply and demand planning, and 5) initiate required actions to accommodate instantaneous deviations from the plan, which is considered an inevitable foundation to determine optimal power supply and demand planning. From that point of view, the latest and historical status, forecasting system performance, plans such as charge and discharge, and other registered actions, are managed within the smart grid digital twin.

Figure D-5-1-1 shows the positioning of the smart grid digital twin in such a power grid system.

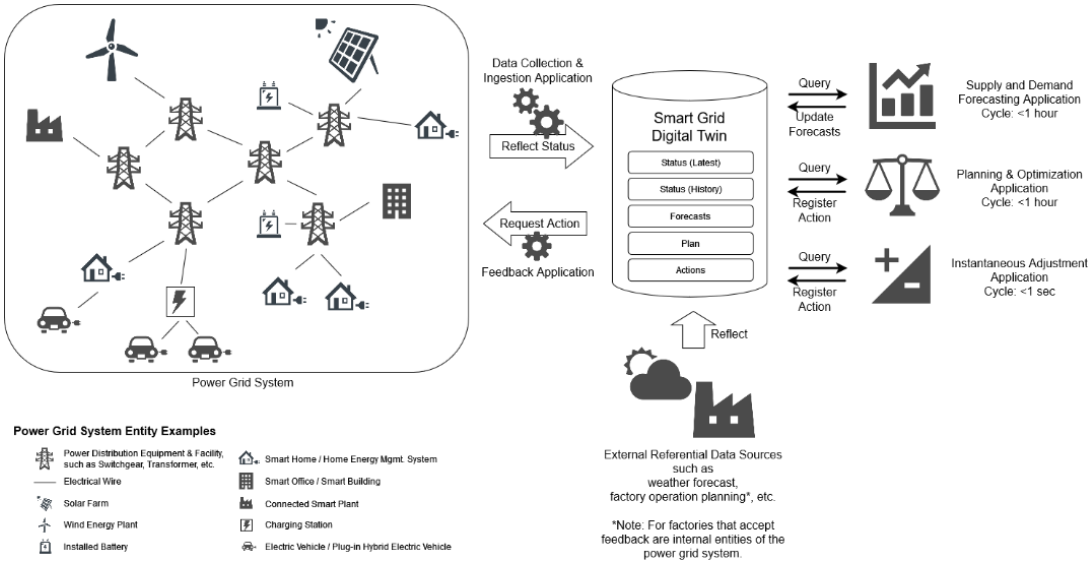

Figure D-5-1-1, An Image of the Smart Grid Digital Twin

D.5.2. Data Management Requirements

First of all, the power grid system is assumed to consist of the following entities:

- Electrical Wires: >1,000,000
- Switchgear Devices and Transformers Facilities: 50,000
- Power plants such as Solar Farm and Wind Energy Plant: 1,000
- Smart Home with Smart Meters: 2,000,000
- Connected Smart Plant: 5,000
- Distributed Energy Storage Resources, such as EV cars: 1,000,000

Also, the data update/ingestion frequency, time interval of records, and data retention period for each information category are assumed in table D-5-2-1:

Table D-5-2-1, Information Categories and their characteristics in the Smart Grid Digital Twin

	Data Update Frequency	Time Interval of Records	Data Retention Period
Latest Status Information	Every second or less	1 second or less	Instantaneous information only
Historical Status Information	Every minute or less	minute or less	10 year or more
Forecast Information	Every hour or less	minute or less	▪ Forecast and plan is made for up to next 48 hours ▪ Forecast and plan old versions are kept 48 hours or more • (if the old versions are kept longer, then the accuracy of the forecast and plan can be evaluated later.)
Plan Information	Every hour or less	minute or less	The same as above
Action Information	Every 100 seconds (assuming 1% of connected entities except for the electrical wires are instructed every second.)	-	10 year or more

With these assumptions, the following requirement must be considered for each information category in the smart grid digital twin:

- Status Information (Latest)
  - More than 4,000,000 entity record updates need to be supported every second or less.
  - Each record may contain hundreds of bytes. (assuming UUID-type primary key is used, and several multi-byte attributes are managed per entity.)
- Status Information (History)
  - Ingestion of more than 4,000,000 entities needs to be supported every 5 minutes or less.

    - More than 5,000,000 records need to be kept for each entity.
    - Each record may contain hundreds of bytes.
- Forecasts & Plan Information
    - As the plan information is created based on the forecasts, and the time interval of records, or granularity of data should be the same (otherwise it makes less sense for the data management burden), the forecasts and plan information of the selected entity is assumed to be managed in the same record.
    - More than 2,880 forecasts-and-plan time-bucket record updates need to be supported every minute for each entity.
    - More than 2,880 forecasts-and-plan versions need to be kept.
    - Each record may contain hundreds of bytes.
- Action Information
    - More than 30,000 action record ingestion need to be supported every second.

Also, the following requirements must be considered for expected digital twin applications:

- Data Collection & Ingestion Application
    - The application needs to be run in a scalable manner so that data can be collected from more than 3 million devices.
    - Collected data needs to be held for more than 1 minute in the application so that not only the latest status information but also the historical status information can be created in the smart grid digital twin.
    - Bulk data ingestion of the latest status information needs to be done with sub-second latency as the latest status information needs to be updated every second or less.
- Supply and Demand Forecasting Application
    - Forecast jobs need to be executed every hour or less, to forecast operational status of 4 million or more entities.
    - The application needs to extract relevant information from the smart grid digital twin in minutes or less so that enough time budget can be allocated to the forecasting application as well as forecast information can be kept fresh.
    - Each entity such as a switchgear in the power grid system has a limited transmission capacity, especially in the upstream direction, therefore, it is necessary to predict the supply and demand balance not only for the entire power grid but also for each point in the power grid.
    - It is necessary to consider the network structure of the power grid when forecasting.
- Planning & Optimization Application
    - Planning jobs need to be executed every hour or less, just after the completion of forecasting application jobs, to update planned controls for 3 million or more entities.
    - The application needs to extract relevant information from the smart grid digital twin in minutes or less so that enough time can be allocated to the planning and optimization application as well as plan information can be kept fresh.
    - It is desired to balance the supply and demand in each point in the power grid, because this minimizes power loss in transmission. Charge/discharge instructions for distributed energy storage resources are determined with this in mind.
    - However, the remaining capacity of each distributed energy storage resource must be maintained at an appropriate level in order to successfully respond to instantaneous power supply and demand disruptions. In particular, the remaining capacity should not be too low.

- In addition, when the amount of power generation becomes excessive, it may become a problem, but in the future, it is expected that such extra power can be absorbed by methods such as using it for hydrogen generation, so it will be less problematic than the case where the amount of power generation is insufficient.

- Instantaneous Adjustment Application
  - Whenever the latest status information is updated, the application should check it against the planned state. If any unacceptable deviation is found, it retrieves the locally relevant information and determines the appropriate instantaneous control in far less than a second.

D.5.3. IDH Solution

To build the proposed smart grid system, the above four different data categories need to be managed and integrated in an effective and efficient manner in the digital twin model, while satisfying the expected application requirements. Based on these requirements, the IDH services will be adopted as shown in figure D-5-3-1.

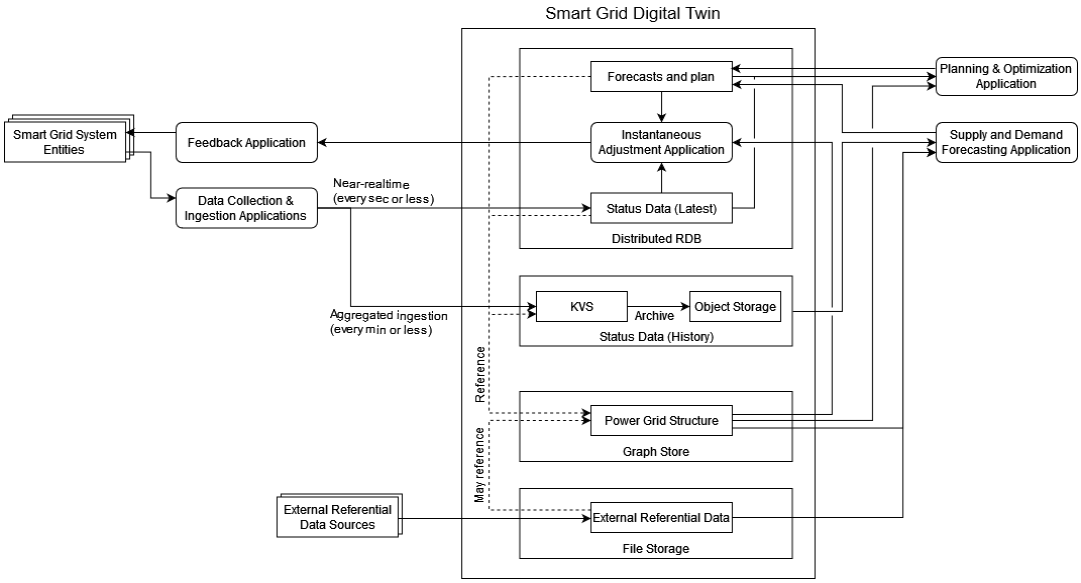

Figure D-5-3-1, IDH Service Mapping to build the Smart Grid Digital Twin

The technical points of each IDH service used for the smart grid digital twin are described below:

- Distributed RDB
  - Data schema needs to be predefined so that data processes can be accelerated for supporting real-time applications. Thus the Distributed RDB is used rather than the KVS in which data schema is not predefined. For this purpose, different data depending on the entities needs to be abstracted.
  - An in-memory-based implementation model may be used as there are millions of records that need to be updated every second.
  - The instantaneous adjustment application may need to be embedded within the Distributed RDB used, so that data processing latency can be minimized by eliminating data movement.
  - The Distributed RDB needs to be connected to the Graph Store through the IOWN OpenAPN or the IDH Converged DB needs to be used to to manage table and graph data, because the embedded real-time application requires such data to determine the appropriate actions when supply-and-demand disruption happens.

- KVS

    - As the different entities generate different status data, a flexible data store is required to accommodate such differences. Therefore, the KVS is used for storing historical status information.
    - When the historical status information needs to be preserved for a longer period of time such as 10 years, then the storage volume becomes so large. Therefore, to reduce the storage cost, the old status data may be moved to cheaper storage model such as Object Storage.
    - To forecast the power supply and demand, the relevant historical status information needs to be extracted and summarized very quickly. To accelerate such a query, microservice applications may work with the KVS, the Distributed File Storage, and/or the Object Storage and process a subset of data in parallel. Also, the data may be converted to the columnar-oriented format when stored, or sub-total may be calculated in advance.
- Object Storage
    - The Object Storage is used to store stale data in a cost-effective manner.
    - By using the query acceleration mechanism of the IDH Object Storage, it is possible to analyze data in Object Storage at a certain speed.
- Distributed File Storage
    - The Distributed File storage is used to import and manage the external referential data within the smart gird digital twin.
    - Unlike Object Storage, the Distributed File Storage can maintain sessions, so it is useful when transferring reference data more frequently and will be beneficial in shortening the forecasting cycle in the future.
- Graph Store
    - The Graph Store is used to represent the power grid system structure.

D.6. Green Twin for the CPS Society Management Use Cases

D.6.1. Use Case Overview

Another example is the so-called Green Twin that monitors and coordinates society’s activities to save energy while increasing their living value. For example, in urban areas, many cars drive to the same place many times to find a parking spot. This wastes energy, emits CO2, and reduces the value of people's lives. What would happen if we could monitor the population’s living spaces, manage the efficiency of their energy use in digital space, and deliver appropriate recommendations to individuals to help them save resources? Obviously, the energy consumption in the real world would be reduced, and population’s quality of life would be enhanced. In addition, the Digital Twin of a neighborhood environment, such as a parking spot, is related to other environments such as building management and public transportation. For example, parking status can be used to assess building activity patterns while still accommodating scheduled events to predict parking availability in the future. We can see that analytics services applied in the digital world are meant for understanding current situations, predicting future situations, and recommending actions. Digital Twin data is shared among twins, and analytics services might relate to multiple Digital Twins. The granularity of the Digital Twins is not fixed; it can be infinitely coarse or fine-grained. Digital Twins interact with each other in the sense of Digital Twin relationships with different granularity, Digital Twin analytics services related to multiple twins, and shared data among Digital Twins. In addition, Digital Twin represents physical twins in the real world. Therefore, the digital twin analytics process needs to have a spatio-temporal understanding of the real-world dimensions. Various IDH services are used to collect data from the real-world, to represent the real-world situation or to build the digital twin, to manage the relationship between the digital twins, and to support various spatio-temporal analyses.

Figure D-6-1-1 shows possible use cases and the relationship between them that are managed under the Green Twin solution framework.

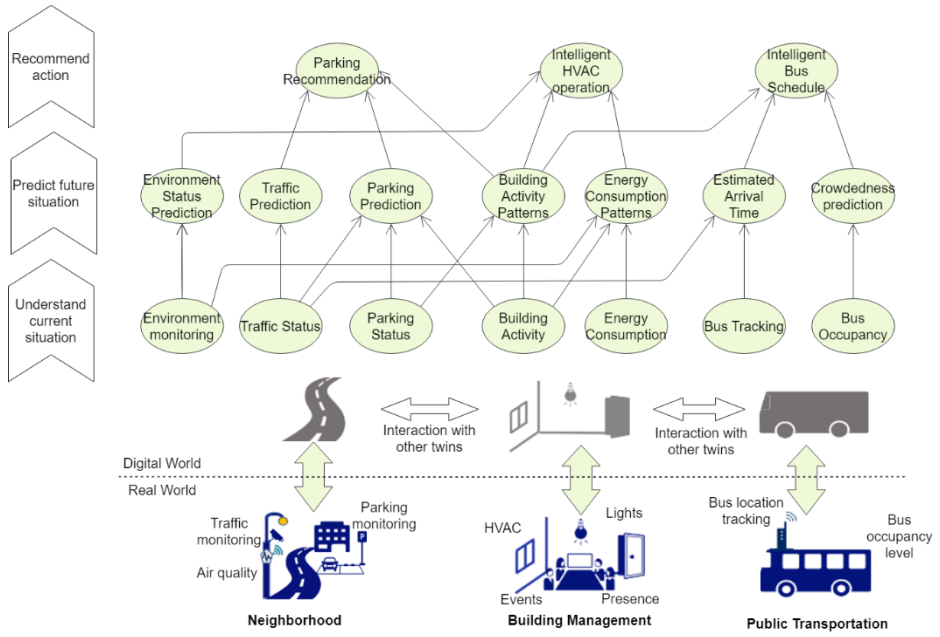

Figure D-6-1-1, Green Twin Use Case

D.6.2. Data Management Requirements

To realize such a Green Twin platform, it will be required to curate, analyze and utilize a large amount of diverse data types. In terms of quantitative analysis, we take as a reference a university building of the campus of the University of Murcia (Spain) (https://www.um.es/web/universidad/mapas/medicina) composed of 6 levels (2 of which are underground), 500 rooms, and 40 hallways. We consider a total of 400 persons per building and 30 vehicles in the proximity of the building.

Table D-6-2-1, Data collection speed of a typical Green Twin use case

Managed Entities	Data Sources (total number of data sources)	Data Collection Speed in total
Smart Building	▪ Installed camera (180/building x 1 building = 180) ▪ Installed meters (700/building x 1 building = 700) • Installed sensors (380,000/building x 1 building = 380,000)	28 Gps
Person	▪ Wearable camera (1/person x 400 persons = 400) • Wearable sensors (10/person x 400 persons = 4,000)	64 Gbps
Vehicle	▪ External camera (8/vehicle x 30 vehicle = 240) ▪ LiDAR sensor (1/vehicle x 30 vehicle = 30) • Other sensors (4/vehicle x 30 vehicle = 120)	544 Gbps

Network Topology	▪ Access network devices, e.g., 5g small cells, Wi-Fi access points, Reconfigurable Intelligent Surfaces (78/network topology x 1 network topology = 78) ▪ Network infrastructure devices, e.g., routers/switches, controllers (13/network topology x 1 network topology = 13) ▪ User devices (3/person x 400 persons = 1,200)	2 Mbps

To utilize this data, there must be multiple functions implemented in the target Green Twin, which are summarized below:

- Digital Twin represents the real world; thus, spatio-temporal characteristics should be considered as native information.
  - Geographic discovery of Digital Twin
  - Type-based discovery of Digital Twin (e.g., request for all *parking slots*)
  - Historical records of Digital Twin
- Digital Twins are related to other Digital Twins without fixed boundaries of granularity:
  - sensors to objects: e.g., *device1* and *device2* sense in *room1*
  - objects to events: e.g., *meeting1* happens in *room1*, *person1* and *person2* are involved in *meeting1*
  - objects to objects: e.g., *room1* is in *building1*, *parkingSlot1* is in front of *building1*, *car1* is parked in *parkingSlot1*
- Heterogeneous data sources:
  - Camera stream linked to a Digital Twin
  - Sensor stream linked to a Digital Twin
  - Static data or semi-statics data (such as building 3D model) linked to a Digital Twin
- Digital Twin framework comprises both data and analytics services. Analytics service might request data synchronously (query-response) and asynchronously (subscription-notification).

D.6.3. IDH Solution

Building a Green Twin solution is very complex since it requires multiple components to manage different kinds of data. An example of a system implementation is given in figure D-6-3-1. The central component of the system is the Context Broker that handles the overlay of the digital twin among different business domains. The links between digital twins and their semantic representation are kept in a Graph Store. An entity linkage processing system discovers semantic links between digital twins based on the data or on semantic descriptions.

Bulky data files such as videos or 3D files are stored in the Object Storage or in the Distributed File Storage and references in the Context Broker. Sensors data flows through the Message Broker and is stored in the KVS. After contextualization and integration following the digital twin semantic model, sensor data is handled in digital twins overlay of the Context Broker. Simulation models are interfaced with the Context Broker to retrieve digital twin data and augments digital twins with simulated outcomes such as future prediction or what-if scenarios. The Green Twin analysis application uses, then, the digital twins that are integrated, contextualized and augmented by simulation for recommendations to humans or actuation systems.

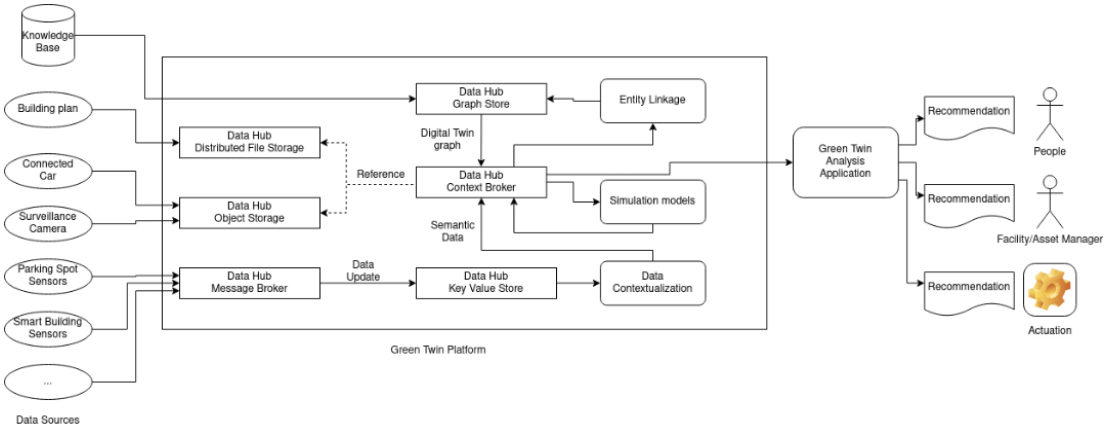

Figure D-6-3-1, IDH Service Mapping to build the Green Twin

D.7. Metaverse Virtual Space for the AIC Entertainment Use Cases

D.7.1. Use Case Overview

The last use case example is the Metaverse. In the Metaverse, many people communicate and interact with each other, enjoy entertainment together, and learn from each other in a virtual space. Also, it can be used to find solutions by imitating production and/or social activities in the real world and performing simulations under different conditions. In other words, future Metaverse solutions will allow us to connect with our family, friends, colleagues, etc. from a distance, or accurately evaluate the value of physical investments before they are made, by reproducing and simulating the real-world situation precisely with sub-centimeter accuracy, etc. In that way, the Metaverse will contribute to creating new values in human society.

Figure D-7-1-1 shows an image of a potential metaverse infrastructure and its usage for multiple use cases.

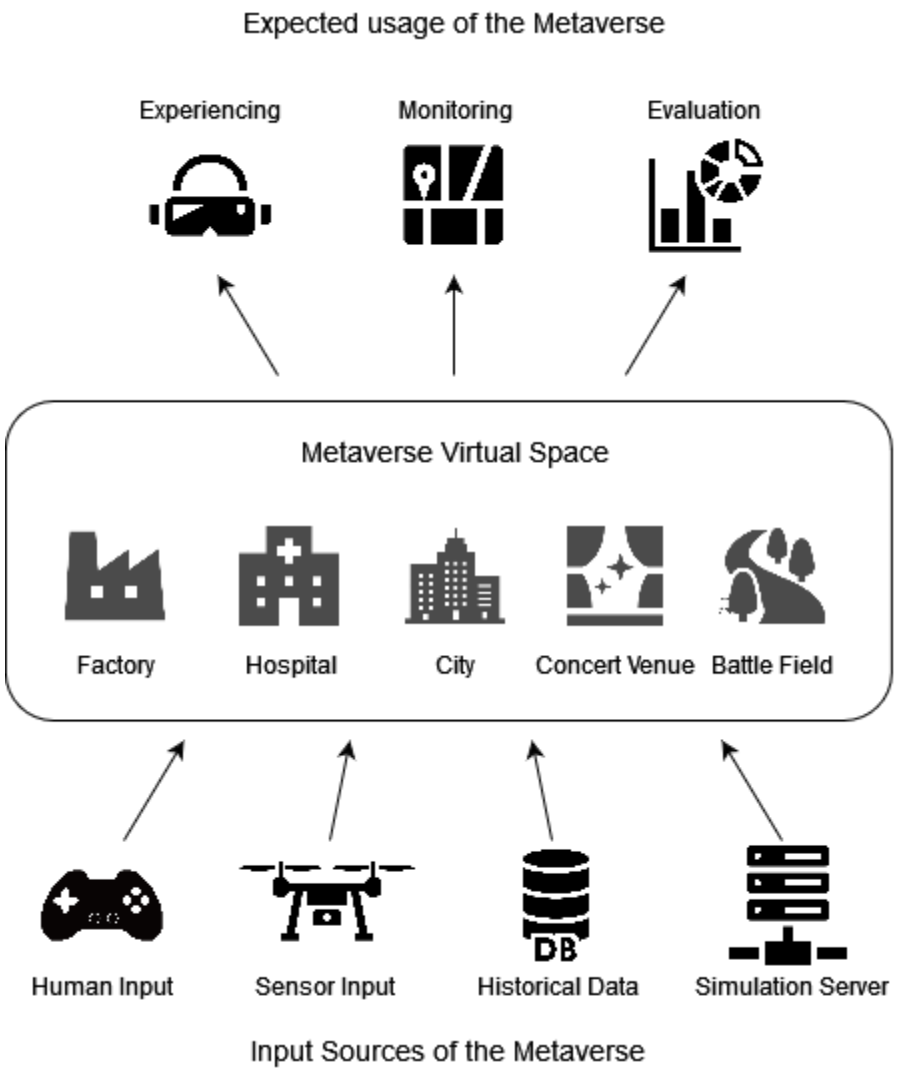

Figure D-7-1-1, An overview on the Metaverse virtual space use cases

D.7.2. Data Management Requirements

There are many challenges in realizing such a Metaverse solution. This is because the Metaverse needs to have the same level of detail as the real world, which means all the phenomena that occur in the real world must be represented as data. Even when targeting limited spaces such as a concert venue or a factory, it is difficult to track thousands or tens of thousands of people, devices, and materials in a cycle of 100 milliseconds or less.

To build an example, a virtual metropolitan city with the following characteristics is used to clarify the data management requirements for the Metaverse.

- The area of the city: 100 square km (10 km x 10 km)
- The number of static objects (building, room, tree, etc.): 1,000,000,000
- The number of semi-static objects (furniture, book, etc.): 1,000,000,000
- The number of dynamic objects (vehicle, helicopter, robot, etc.): 1,000,000
- The number of living people (active avatars): 1,000,000

In order to express such a virtual metropolitan city as data, static map data is needed as a foundation. It represents terrain and static building and room structures. The size of this static data varies depending on the resolution, etc., but the data is rarely updated.

Then, the semi-static objects also need to be managed. These semi-static objects also move and change their states because of human activities. In this analysis, the number of semi-static objects that are moving or changing state is assumed to be the same as the number of people, i.e., 1,000,000.

Also, the dynamic objects, such as virtual vehicles, are managed. In this analysis, the number of such dynamic objects that are actually used and moving is assumed to be 10% of the total number of the dynamic objects, i.e., 100,000.

Lastly, people, or avatars of people, are also managed. Basically, all people are always doing some kind of activity, so their positions, movements, and states are always expressed as data.

Under such assumptions, if position, posture, and state of these objects are managed every 100 milliseconds, the data management requirements for this virtual metropolitan city will be summarized in table D-7-2-1.

Table D-7-2-1, Data management requirements of a virtual metropolitan city

	The number of total records	Record structure characteristic	Update frequency
Static Map	1 billion	incl. ID, type, position, shape, color, etc.	Not many
Semi-static objects	1 billion	incl. ID, type, position, shape, color, etc.	10 million per second
Dynamic objects	1 million	incl. ID, type, position, shape, color, etc.	1 million per second
People (Avatars of people)	1 million	incl. ID, position, posture, clothes, etc.	10 million per second

D.7.3. IDH Solution

As described in section D.7.2, data management requirements of the Metaverse are incredibly daunting. With today's database technology, it would be very difficult to support millions of data updates per second and more lookup operations when record sizes exceed several kilobytes as they include posture, shape, etc.

Therefore, the IOWN Global Forum plans to develop an IDH-based solution for realizing such a Metaverse virtual space. With IDH-based solutions, multiple servers, CPUs, memories, etc. are interconnected via IOWN OpenAPN, and it becomes possible to process the large amounts of data required for the Metaverse.

Figure D-7-3-1 shows one possible configuration of such an IDH-based Metaverse virtual space solution.

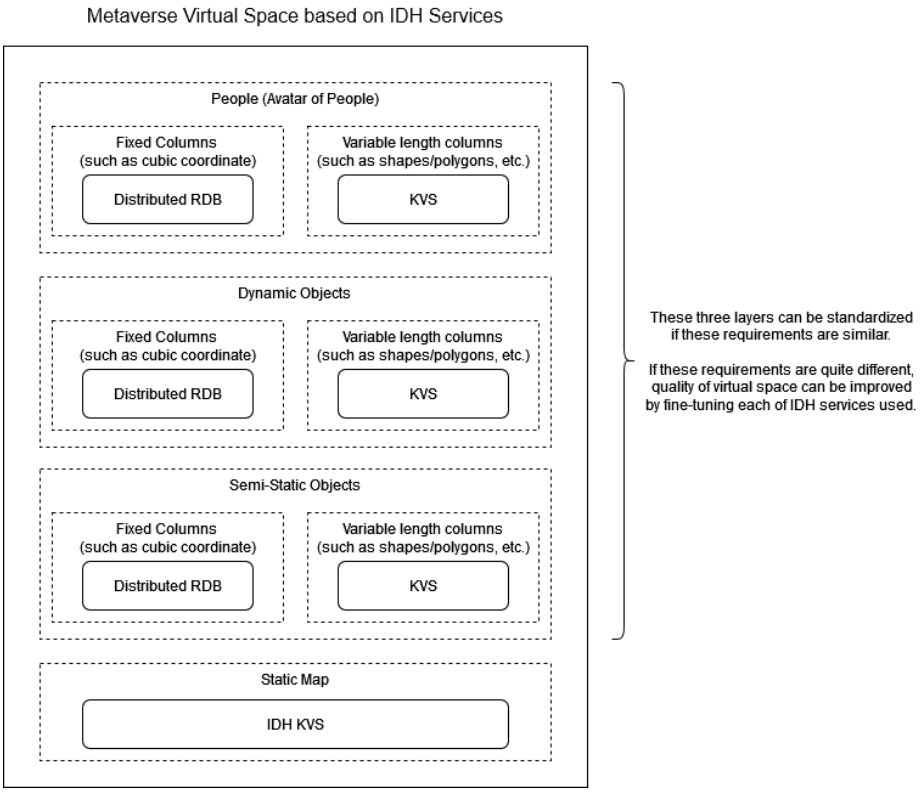

Figure D-7-3-1, IDH Service Mapping to build the Metaverse virtual space

Based on such a structural assumption, it would also be required to make each IDH service scalable to meet the data management requirements described in section A.7.2. When designing the IDH services for such scalability, it is important to ensure that the allocation of data and workload to each server is almost even, and inter-server data processing and movement, which are considered overhead, are minimized.

As a good design candidate for such distribution, there is a method to divide the virtual space into smaller unit spaces and use them to control the allocation of data and workloads to each server, assuming that participants in the virtual space, i.e. avatars, are distributed in the same way as in the real space due to seat allocation and physical collision detection. When applying such a design, the following points must be considered:

- Allocation of unit spaces to each server:

  Each server needs to be allocated an equal number of multiple unit spaces. This is because the workload on each unit space is expected to be very different, so if only one or a few unit spaces are allocated to each server, then the server workload will be very unbalanced.

- Size of unit space:

  Neither too little nor too large is good. This is because if it is too small, there will be more cross-unit-space queries, and more people and objects that move across the unit space boundaries, and therefore data processing and movement overhead will increase. If it is too large, the number of unit spaces allocated to each server becomes less, which results in uneven server workload.

- Dynamic scalability

  It is desirable that the IDH service infrastructure be able to scale according to the increase or decrease in the number of people (avatars) in the virtual space. For that reason, it is important to be able to dynamically change the number of unit space allocations. A simple method is to use a consistent hashing algorithm to rebalance data and workload across the servers. However, in that case, a certain imbalance state may occur. To achieve a more accurate balance, it is also possible to monitor the amount of data and workload

for each unit space and determine the reassignment to the server based on the monitoring results and calculations. In the latter case, it is necessary to set up a mapping management server.

Figure D-7-3-2 shows an image of a scalable platform for the Metaverse virtual space when a hexagon is used as a unit space.

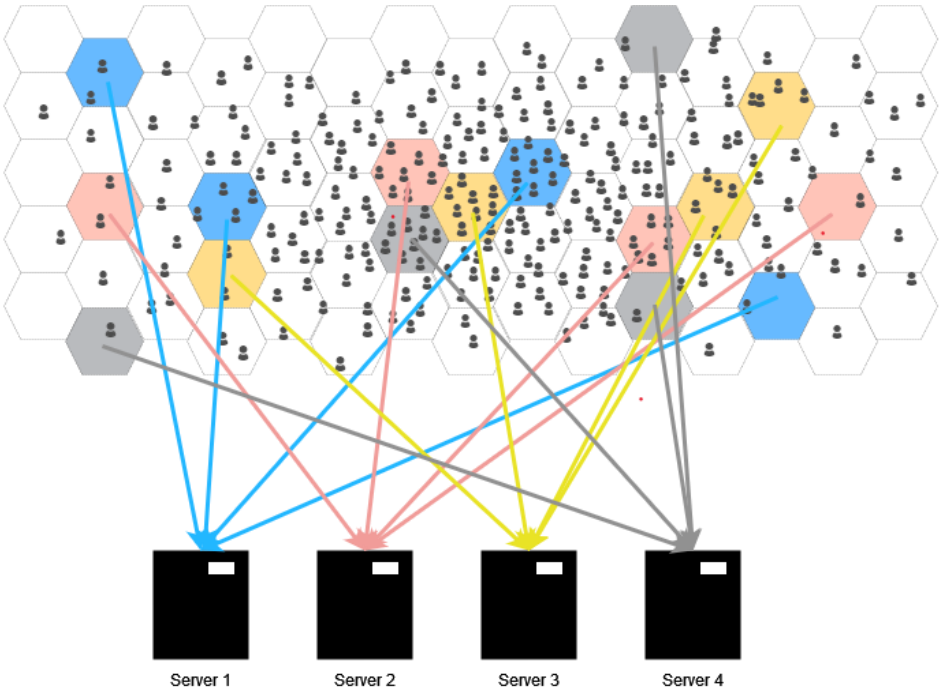

Figure D-7-3-2, Scalable IDH Service design

…

History

Revision	Release Date	Summary of Changes
1.0	2022/2/1	Initial version
2.0	2023/07/20	Chapter 1 ▪ More detailed explanation on the IOWN Global Forum Data Hub solution was added. Chapter 2 ▪ An analysis of a wide range of use cases that IOWN Global Forum had envisioned was added. Chapter 3 ▪ Service class details were moved to Annex A, and only high-level explanations were left in Chapter 3. Chapter 4 ▪ Today's implementation models of each service class were moved to Annex B. ▪ Instead, a summary of common gaps across today's implementation models was provided. Chapter 5 ▪ Expected implementation models of each IDH service class were moved to Annex C. ▪ An expected common architecture structure across IDH services was added. ▪ Expected functional architecture options were described and discussed. Chapter 6 ▪ IDH use case descriptions were all moved to Annex D. ▪ Instead, a high-level explanation of the value of the IDH solution was added.