StreamStorage: High-throughput and Scalable Storage Technology for Streaming Data

● Munenori Maeda ● Toshihiro Ozawa

Real-time analytical processing (RTAP) of vast amounts of time-series data from sensors, server logs, and other sources has come to be widely used in recent years. To make thorough use of such streaming data, it is essential that the same data be analyzed iteratively from diverse viewpoints, which has increased the need for loss-free storage and data reconstruction functions for large volumes of data. Since individual items of data targeted for storage are generally small in size and repeatedly received from multiple sources, it is difficult for existing storage systems to simultaneously satisfy throughput and capacity demands. Additionally, high-speed access optimized for the chronologically ordered data characteristic of streaming data has not yet been achieved for such vast amounts of data. StreamStorage developed by Fujitsu Laboratories is a new storage technology that manages time-series data in units of streams, enabling high-speed storage and data reconstruction. It achieves high throughput and scalability by using distributed storage technology in which streaming data is partitioned into blocks and data is input and output in a parallel and asynchronous process. This paper presents an overview of the StreamStorage architecture and an application example.

1. Introduction

Real-time analytical processing (RTAP) of vast amounts of streaming data from sensors, server logs, and other sources has come to be widely used in a variety of fields including smart grids, wireless sensor networks, and algorithmic trading.

Conventional RTAP systems use databases in the form of a relational database (RDB) or Hadoop Distributed File System (HDFS) for storing streaming data and analysis results, taking into account links with various types of analysis applications. These types of storage, however, have difficulty keeping up with high throughput and capacity demands.

If vast amounts of streaming data can be stored in full without loss, it should be possible to use data in ways that go beyond conventional real-time analysis:

- Analysis of the same data stream from different viewpoints, enabling cycles of hypothesis testing that can lead to new findings
- Flexible setting of the analysis technique and the range of target data, enabling the skipping of analysis up to a certain point in time, partial changing of the analysis technique after winding back the clock to a previous point in time, etc.

We can therefore consider that there are many needs to be satisfied in storing vast amounts of streaming data.

One of main issues in storing vast amounts of streaming data is how to achieve high write throughput. An item of data targeted for RTAP is from several tens of bytes to tens of thousands of bytes, which is extremely small compared to the I/O size (several MB) for eliciting good performance from hard disk drives. It is therefore essential that appropriate schemes be used, such as grouping multiple items of data for input/output and accessing them in a parallel and asynchronous way.

Two requirements are proposed for facilitating ease of use:

- Handle semantically related streaming data as a single, logical entity such as a file or RDB table.
- Enable an application program to access stream data sequentially in chronological order without having to be particularly aware of the order.

For data storage and reconstruction to be provided as a service, it must be possible to increase or decrease the performance of that service by changing the amount of resources on the service platform (scalability). This will enable the service to be used continuously from when it is still a "small start" service to when it has expanded into a large-scale service, with the corresponding changes in the volume and frequency of accesses.

StreamStorage, developed by Fujitsu Laboratories, is a storage technology with a natural application program interface (API) that achieves scalability and high throughput. It handles data streams as units of storage and accesses each piece of data inside a stream in chronological order.

This paper first outlines the architecture of StreamStorage and then describes the linking of StreamStorage with complex event processing (CEP) as an application example.

2. Architecture

As the name implies, streaming data in StreamStorage is managed in logical units called "streams," which correspond to files in conventional storage systems. The content of a stream is arranged in some order such as the order of transmission or the order of reception.

A StreamStorage user allocates, from among the data received from various sources, a single stream to a group of data that the user would like to treat as a series of information items in some order. For example, data-transmission sensors can be divided into groups, and a stream can be allocated to each group, or received data can be divided into streams in accordance with the content of that data. In short, a variety of usage formats are possible. An example of a StreamStorage usage format as seen from the application side is shown in **Figure 1**.

StreamStorage, as seen from the application side, must be able to do three things:

- Store and reconstruct small-size items of data at high throughput
- Enable a stream to be simultaneously accessed from multiple applications (users)
- Raise performance by increasing the number of servers (scale out) so that huge volumes of data can be handled.

Three guiding principles on architecture design were adopted to meet these functional requirements:

1) Data shall be recorded in parallel and asynchronously without completely ordering the data so that the connection times with sources can be shortened and the number of simultaneous connections can be increased.
2) Small-size items of data shall be buffered in memory, and I/O size shall be increased to raise throughput.
3) It shall be possible to flexibly increase or decrease storage capacity and performance.

Principles 1) and 2) mean that the logical (chronological) order of streaming data is separated from the physical storage locations and storage order. This means that the order of streaming data must be preserved within StreamStorage when each data item in a

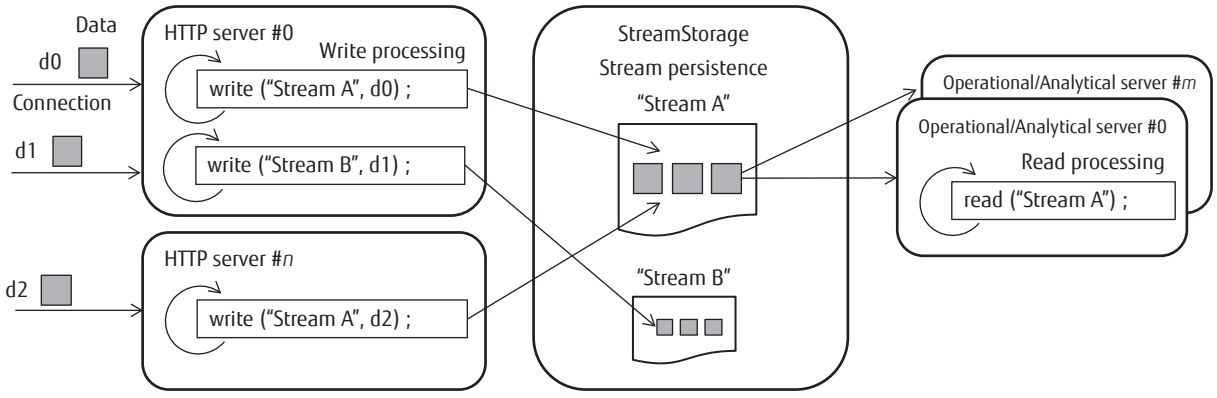

Figure 1
StreamStorage usage format as seen from application side.

stream is read by an application program.

Principle 3) means that a storage infrastructure capable of scale out is required and that a distributed object storage system capable of high throughput and large data capacity is needed for such an infrastructure.

The StreamStorage architecture consists of the following elements (**Figure 2**).

1) StreamStorage client

The StreamStorage client is a software library that enables the HTTP servers and operational/analytical servers to access StreamStorage. It provides a write buffer for temporarily saving streaming data. When the buffer becomes full, the buffered data are integrated and transferred in bulk to a storage proxy and the buffer is flushed. This set of data is called an event block. When a stream is subsequently needed, it is reconstructed from multiple event blocks and saved in a read buffer. The reconstruction mechanism is explained in detail in the following section.

2) Metadata server

The metadata server mainly manages the information needed to reconstruct the stream. Specifically, it associates a stream name and a time interval with the event blocks that comprise the stream during that interval, enabling it to respond to an inquiry from a StreamStorage client.

3) Storage proxy

Positioned on a connecting layer between storage nodes and the StreamStorage client, the storage proxy routes access requests from the client to appropriate storage nodes.

4) Storage nodes

The storage node element is a distributed key-value store (KVS) made up of commodity-grade servers. It stores event blocks in a redundant configuration, such as data replication.

Constituent elements 1)–4) achieve high availability through data redundancy and standby-system switching (failover function) to prevent breakdown due

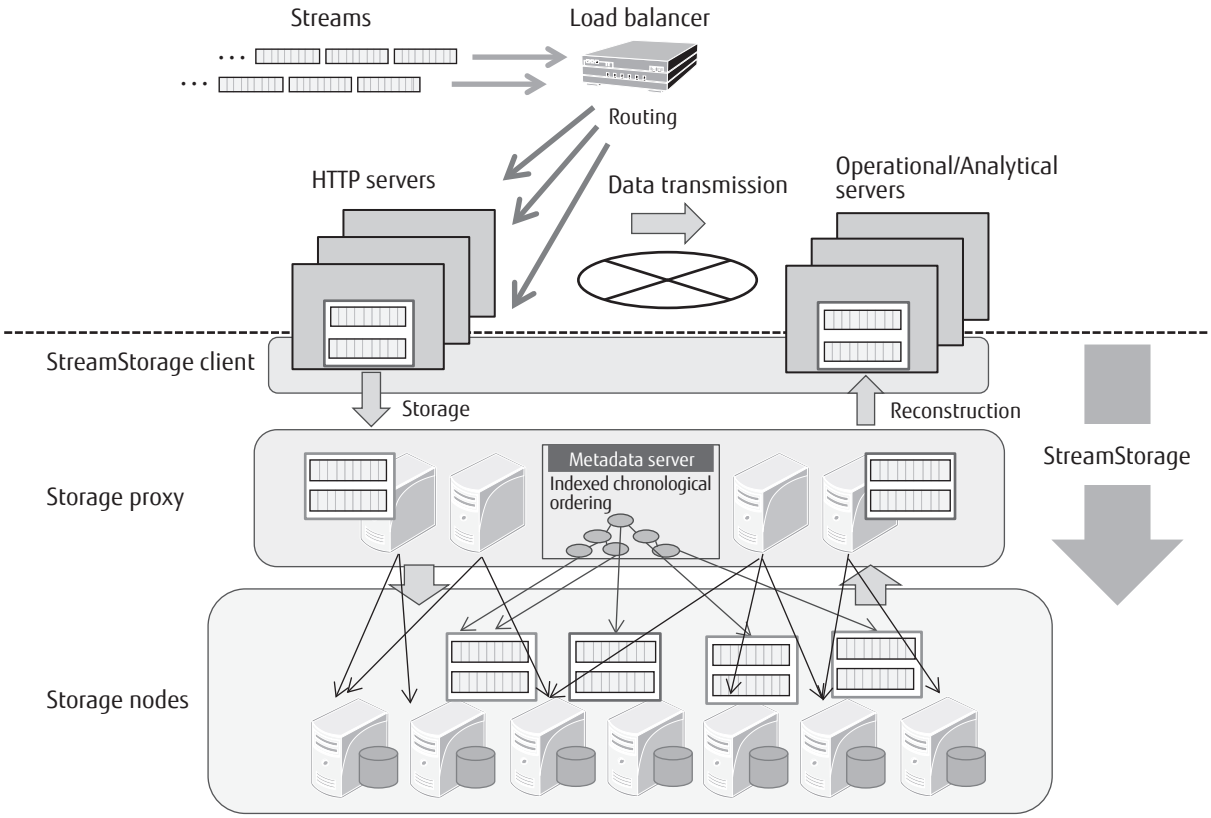

Figure 2
StreamStorage architecture.

to hardware or software faults.

3. Stream reconstruction mechanism

Reconstructing a data stream that has been divided into event blocks and stored as such requires that event blocks within the specified time interval be read out and that data within those blocks be arranged in their correct order. This is accomplished by using the stream reconstruction procedure (**Figure 3**).

1) The StreamStorage client specifies a time interval for reconstruction and sends an inquiry to the metadata server [1) in Figure 3].
2) The metadata server looks up the event blocks overlapping the specified time interval and returns the names of those blocks [2) in Figure 3].
3) The StreamStorage client sends read requests to the storage proxy specifying the names of those event blocks [3) in Figure 3].
4) The storage proxy routes a read request to an appropriate storage node, which reads out and returns the specified event block [4) in Figure 3].
5) The StreamStorage client merges the event blocks read out and returned by the storage nodes [5) in Figure 3] into one stream. Since the data within each event block has already been chronologically ordered upon storage, the merging process has much lower cost than a total sort.

Several practical mechanisms have been developed to achieve data reconstruction:

1) Preloading (smooth reconstruction)

Event blocks from storage nodes are read out before that data is actually needed to make data read time (latency) more uniform.

2) Incremental stream reconstruction

The buffer used by the StreamStorage client for reconstruction is limited in size. To prevent memory depletion during lengthy reconstruction, the specified interval is first divided into appropriately short subintervals, and then the stream is incrementally reconstructed, starting from the oldest subinterval.

3) Data filter

To reduce the amount of data transferred over the network and the merge processing load, screening rules are specified with respect to the data in step 3) of the above reconstruction procedure. The storage nodes then return event blocks that contain only data

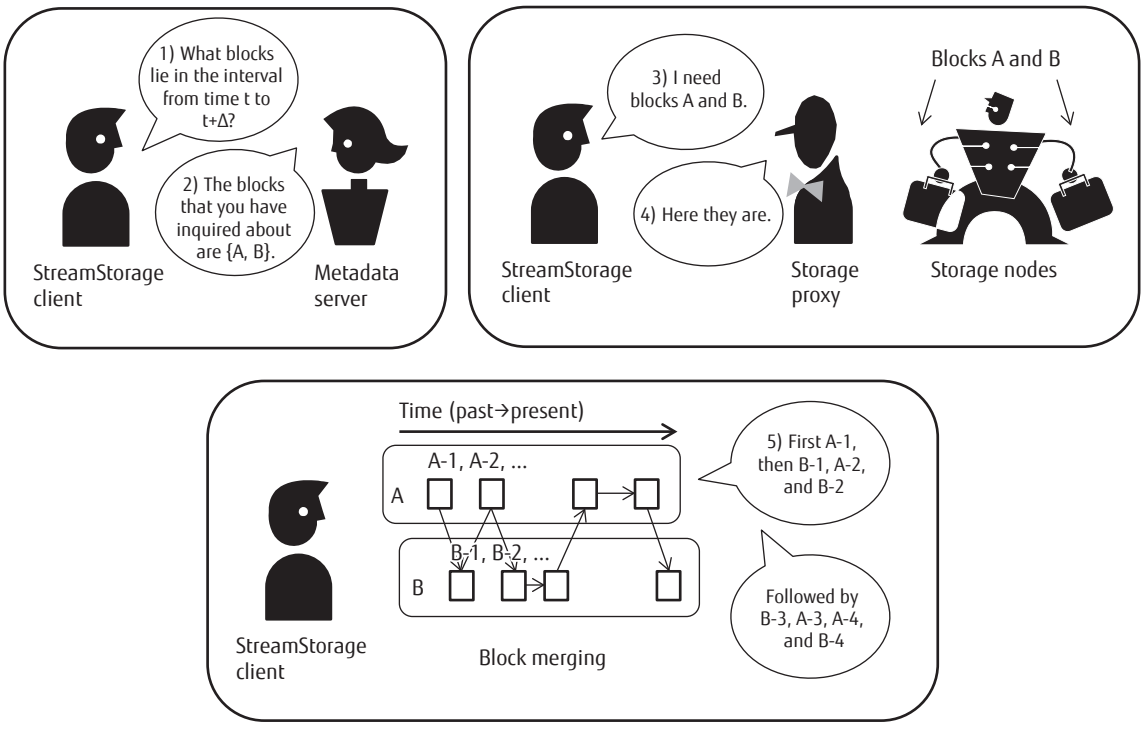

Figure 3
Stream reconstruction procedure.

that pass through that filter. This mechanism is used to achieve elastic-parallelism reconstruction described in the next section.

4. Application example of StreamStorage

We conducted a trial in which StreamStorage was coordinated with a CEP system, which analyzes vast amounts of streaming data and draws inferences at high speed in accordance with predefined rules. This trial was performed as part of the Project for the Program to develop and demonstrate basic technology for next-generation high-reliability, energy-saving IT equipment[1),2)] supported by the Ministry of Economy, Trade and Industry (METI). In this trial, we used two distinctive features of StreamStorage—chase reconstruction and elastic-parallelism reconstruction—to achieve high-availability, i.e., shorten downtime, of a system.

1) Chase reconstruction

This function performs reconstruction from some point in the past to the present while storing received data. By making the speed of reconstruction greater than the speed of storage, reconstruction eventually catches up with storage in the present and then terminates at that point in time. Then, the latest received data is passed directly to the operational or analytical application servers instead of being stored and then reconstructed.

2) Elastic-parallelism reconstruction

This function splits a reconstructed stream into substreams and then distributes them across multiple application servers to increase total throughput. The number of substreams is variable and determines the degree of parallelism that the application servers can provide at runtime. A rule is specified for creating a possible remainder set by dividing the degree of parallelism at the time of reconstruction by that at runtime. This rule acts as a data filter.

Applying a CEP rule that determines output over a certain range of streaming data from some point in the past to the present is a routine practice. Consequently, if a hardware or software fault should temporarily interrupt operations on a CEP server in which such a rule is being applied, there will be some period during which erroneous or less-than-accurate output will continue to be produced even if the server is rebooted. This, in effect, will make it difficult for operational and analytical applications using CEP output to continue operations during that period.

StreamStorage, however, makes it possible to return to the time during which output following the fault occurrence was affected and to perform chase reconstruction from that point so that operations can continue without the operational and analytical applications having to take any special countermeasures. In addition, the duration of chase reconstruction can be shortened by increasing parallelism at the time of reconstruction and performing elastic-parallelism reconstruction.

A fiscal year 2011 commissioned report[2)] stated that the application of chase reconstruction and elastic-parallelism reconstruction to CEP system used for monitoring the water levels of rivers has been shown to be effective.

5. Conclusion

This paper introduced StreamStorage as a storage technology targeting streaming data. StreamStorage enables scalable, high-throughput storage and reconstruction of large numbers of data sets that are individually small in size. It features flexible reconstruction functions including chase reconstruction and elastic-parallelism reconstruction.

Development is still in progress to provide sufficient quality as a component of FUJITSU Software Interstage,[3)] a business application platform.

This research was supported in part by the Ministry of Economy, Trade and Industry's Project for the Program to develop and demonstrate basic technology for next-generation high-reliability, energy-saving IT equipment for FY2010 and FY2011.

References

1) Ministry of Economy, Trade and Industry: FY2010 Work Report on Program to develop and demonstrate basic technology for next-generation high-reliability, energy-saving IT equipment, 2011, pp. 187–276 (in Japanese). *http://www.meti.go.jp/policy/mono_info_service/joho/cloud/2010/02.pdf*

2) Ministry of Economy, Trade and Industry: FY2011 Work Report on Program to develop and demonstrate basic technology for next-generation high-reliability, energy-saving IT equipment, 2012, pp. 173–255 (in Japanese). *http://www.meti.go.jp/policy/mono_info_service/joho/cloud/2011/02.pdf*

3) Fujitsu: FUJITSU Software Interstage: Helping Sense and Respond to Business Changes.
http://www.fujitsu.com/global/services/software/interstage/

Munenori Maeda
Fujitsu Laboratories Ltd.
Mr. Maeda is engaged in research of distributed storage systems.

Toshihiro Ozawa
Fujitsu Laboratories Ltd.
Mr. Ozawa is engaged in research of distributed storage systems.