Journal of Physics: Conference Series **219** (2010) 022012 doi:10.1088/1742-6596/219/2/022012

A common real time framework for SuperKEKB and Hyper Suprime-Cam at Subaru telescope

S Lee[1], R Itoh[2], N Katayama[2], H Furusawa[3], H Aihara[4] and S Mineo[4]

[1]Department of Physics, Korea University, 1 Anam-dong, Seongbuk-gu, Seoul, Korea
[2]IPNS, KEK, 1-1 Oho, Tsukuba, Ibaraki, Japan
[3]National Astronomical Observatory of Japan, National Institutes of Natural Sciences, 2-21-1 Osawa, Mitaka, Tokyo, Japan
[4]Department of Physics, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo, Japan

E-mail: shlee@hep.korea.ac.kr

Abstract. The real time data analysis at next generation experiments is a challenge because of their enormous data rate and size. The SuperKEKB experiment, the upgraded Belle experiment, requires to process 100 times larger data of current one. The offline-level data analysis is necessary in the HLT farm for the efficient data reduction. The real time processing of huge data is also the key at the planned dark energy survey using the Subaru telescope. The main camera for the survey called Hyper Suprime-Cam consists of 100 CCDs with 8 mega pixels each, and the total data size is expected to become comparable with that of SuperKEKB. The online tuning of measurement parameters is being planned by the real time processing, which was done empirically in the past. We started a joint development of the real time framework to be shared both by SuperKEKB and Hyper Suprime-Cam. The parallel processing technique is widely adopted in the framework design to utilize a huge number of network-connected PCs with multi-core CPUs. The parallel processing is performed not only in the trivial event-by-event manner, but also in the pipeline of the software modules which are dynamically placed over the distributed computing nodes. The object data flow in the framework is realized by the object serializing technique with the object persistency. On-the-fly collection of histograms and N-tuples is supported for the run-time monitoring. The detailed design and the development status of the framework is presented.

1. Introduction

There are upgrade plans for both of the Belle [1][2] and the Suprime-Cam [3] experiment. The SuperKEKB is the upgrade of the Belle experiment which is performed at KEK in Japan and its purpose is observation and better understanding of CP violation in B meson decay in asymmetric e^+e^- collision. The planned luminosity of the SuperKEKB is $8 \times 10^{36} cm^{-2} s^{-1}$ so that will generate 50 billion $B\bar{B}$ pairs. The data from the SuperKEKB is 100 times larger than current experiment and the computing for the upgraded experiment is a technological challenge. The online data have to be recorded at a speed of 250 MB/sec after online reconstruction and reduction amounting to the data size of 5 PB/year [4]. The size of each event data will be about 250 KB, that is, about 2 GB data will be produced for every second.

The Hyper Suprime-Cam (HSC) [5] is the upgrade of the Suprime-Cam (SC) experiment which is performed at Subaru telescope in Hawaii. The purpose of the HSC is searching for dark energy. The HSC is next generation CCD camera for Subaru telescope and it will replace the

SC in 2011. In the HSC, 110 CCDs are arranged in grid and each of them take a shot for every 5-20 minutes including the determination of exposure of next shot. Each CCD has 880 million pixels and the data rate for HSC is about 2Gbps at max. In addition the size of each shot is about 1.5 GB, that is, 1.5 GB data will be produced for every 5 minutes and the data should be processed in 20 seconds to determine the exposure of next shot. The data produced by the HSC is 10 times larger than data of the SC and comparable with data of SuperKEKB.

2. Motivation

One of the motivations of development of a framework is larger amount of data of upgraded experiments. It means more powerful and efficient framework is required for the next generation experiments. Between two experiments, the SuperKEKB and the HSC, there are many common interests in data processing. Both of them require real time and on-demand data processing. There is a strong motivation of joint work between the SuperKEKB and the HSC for development of new framework. The next section covers the software pipeline architecture as an optimal solution for them. The current framework of the Belle experiment [8] and the SC has very complicated and old-fashioned structure and has limitations for the new experiment thus we decided to develop a new framework. To satisfy the requirements of new experiments, these limitations should be overcome. The new framework will provide interfaces to not only classical computing resources such as SMP and network clusters but also new computing resources such as GRID and cloud computing which are not supported by current framework.

3. Requirements

As introduced earlier, the software pipeline architecture is very useful for the two experiments. It provides a modular structure so that it makes user-supplied analysis modules be dynamic-linkable and enables flexible execution control of modules. The modules which are processed in the software pipeline architecture are built as shared objects. The software pipeline can perform data processing in real time and it also provides on-demand module pipelining.

Another requirement is object persistency. ROOT [6] is widely used in experimental particle physics and AstroROOT [7], a ROOT package, is preferred to analyze data in astronomy. Data handling of those two tools are based on object-oriented way. In the current Belle experiment, home-grown data format [8] is being used and the data format was not designed in object-oriented way. Therefore it is natural that we design and implement the new framework in object-oriented way. The object persistency gives an advantage in data exchange as well. The new framework should support old data format as a legacy interface.

The parallel processing is also an important requirement. New experiments require massive data processing so that the parallel processing is a key point of maximizing performance of

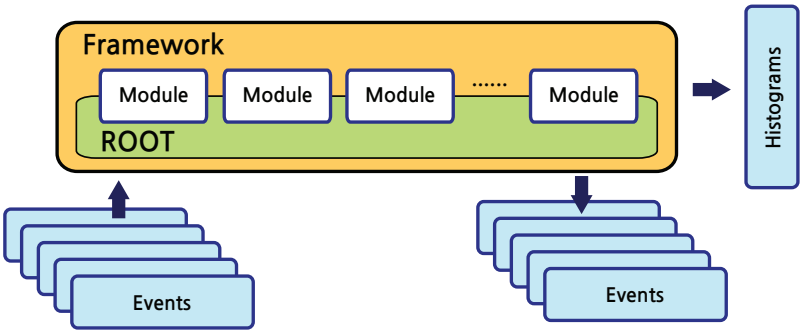


Figure 1. The software pipeline of the framework

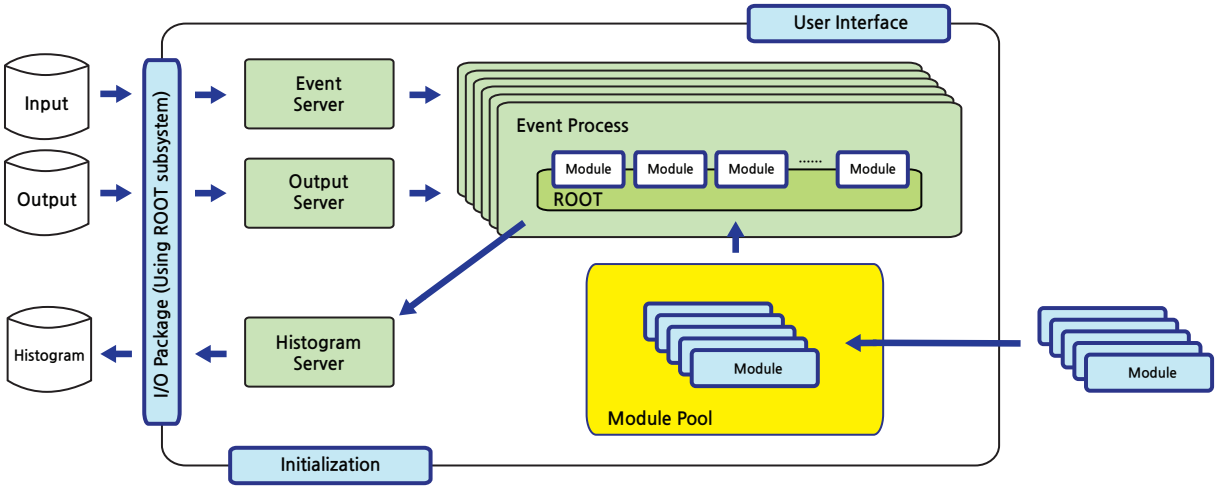


Figure 2. The entire architecture of the framework

various computing resources. In the parallel processing scheme, not only data but also modules should be processed in parallel so that module pipelining is enabled over network. And it also supports various computing resources such as SMP, network clusters and computing grid and provides consistent interface to those computing resources. The parallel processing scheme should have versatile and transparent processing layers so that provides convenient way to manage it. Finally, the framework should provide integrated environment. The framework will be used in different targets such as data acquisition (DAQ) system including readout and high level trigger (HLT), data production, Monte Carlo production and user analysis. Although the usages of targets are little different, the framework should be unified one for all cases. It also provides integrated interface to end-users. The integrated interface makes end-users perform their analysis without any boundary between DAQ and offline analysis so that provides a seamless architecture between them. In addition, integrated interfaces provides consistent interface to end-users no matter what computing resource is used for parallel processing.

4. Architecture

The pipeline architecture is shown in figure 1. Event data are processed by a set of modules sequentially. These modules work in ROOT-based environment.

The entire architecture of the framework is shown in figure 2. The I/O package reads and writes data from ROOT file. The event server distributes data to multiple event processes for the parallel processing. Each event process receives data from the event server and processes them on the software pipeline. Dynamic-linkable modules are placed on module pool so that event process picks up the required modules from the module pool. After processing, event processes send data to the output server back and it makes them as a output (output data or histograms). Figure 3 and 4 shows more details of real time pipelining in SuperKEKB and HSC, respectively.

5. Key Functionalities

The key functionalities of the framework are the object persistency and the parallel processing. The object persistency provides convenient way to handle data and an interface to ROOT. The parallel processing is combined with the object persistency so that provides transparent and versatile functionalities.

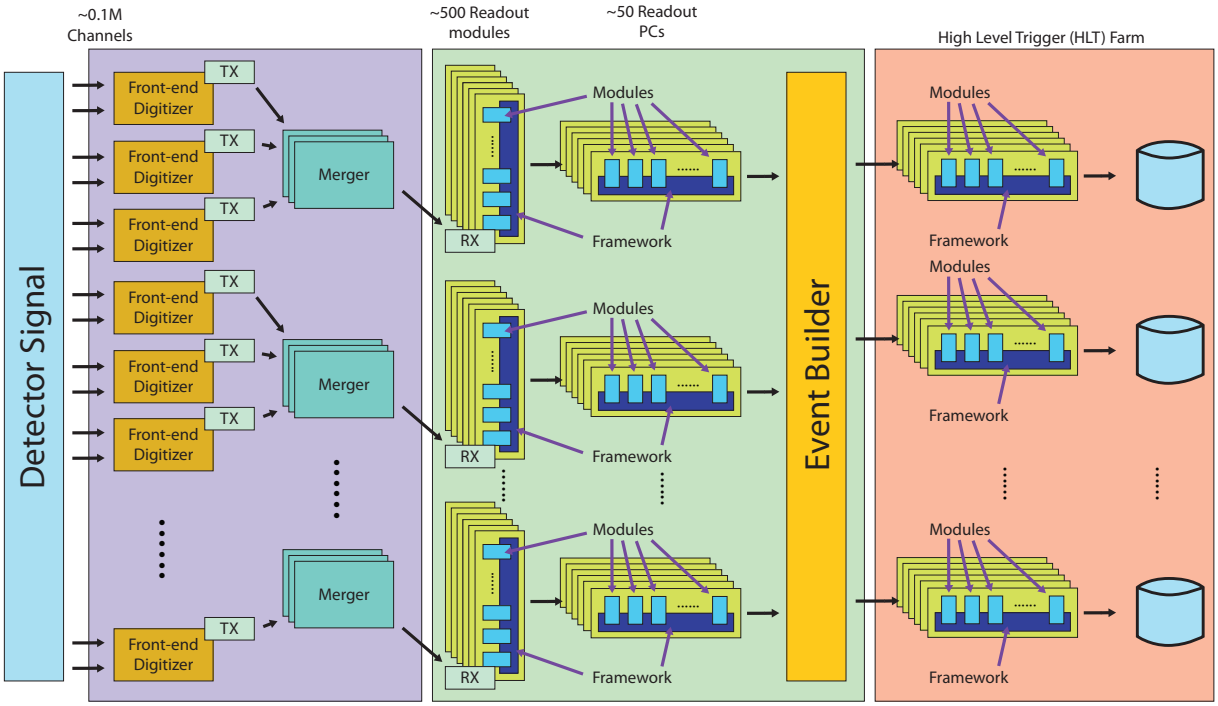


Figure 3. The real time pipelining in the SuperKEKB. The framework is engaged in readout modules, readout PCs and HLT. TX and RX are components which involve in transmitting and receiving data, respectively.

5.1. Object Persistency

The object persistent data handling provides convenience to both of developers and end-users. In our framework, all data are regarded as objects. We define two base object for data handling. The one is the BelleEvent object and the other one is the EvtMessage object. The BelleEvent object is a basic unit of event data at user level and it is derived from the TObject class of ROOT. Event data are encapsulated in the BelleEvent object so that contains information of particles and events and provides convenient functions to handle those information. It can also contain any other objects which are defined by user. Thus the BelleEvent object is very flexible, versatile, and easy to be extended. The EvtMessage object is an object which is used actually inside the framework for transferring data. The EvtMessage object is derived from the TMessage class of ROOT so that provides object deserialization which is implemented as a member function of the TMessage class. The EvtMessage object contains data serialized from the BelleEvent object and information of the data. The BelleEvent object itself is not convenient to be transmitted so that we define additional class of the EvtMessage for data exchange. Figure 5 shows the conceptual view of our object persistency inside the framework.

Figure 6 shows how data flow and in which way components are involved inside the framework with the object persistency. The ROOT I/O component reads data from input ROOT file. And the event server encapsulates every single event to the BelleEvent object and converts it into the EvtMessage object. If the event server is located in separated node from data reading, it receives the EvtMessage object itself from network. This is an idea of the parallelization over network of our framework. The converted EvtMessage object is serialized and is sent to the FIFO such as shared memory. The event processes receive the serialized data from the FIFO to process them. Received data is restored to the EvtMessage object and is converted to the BelleEvent

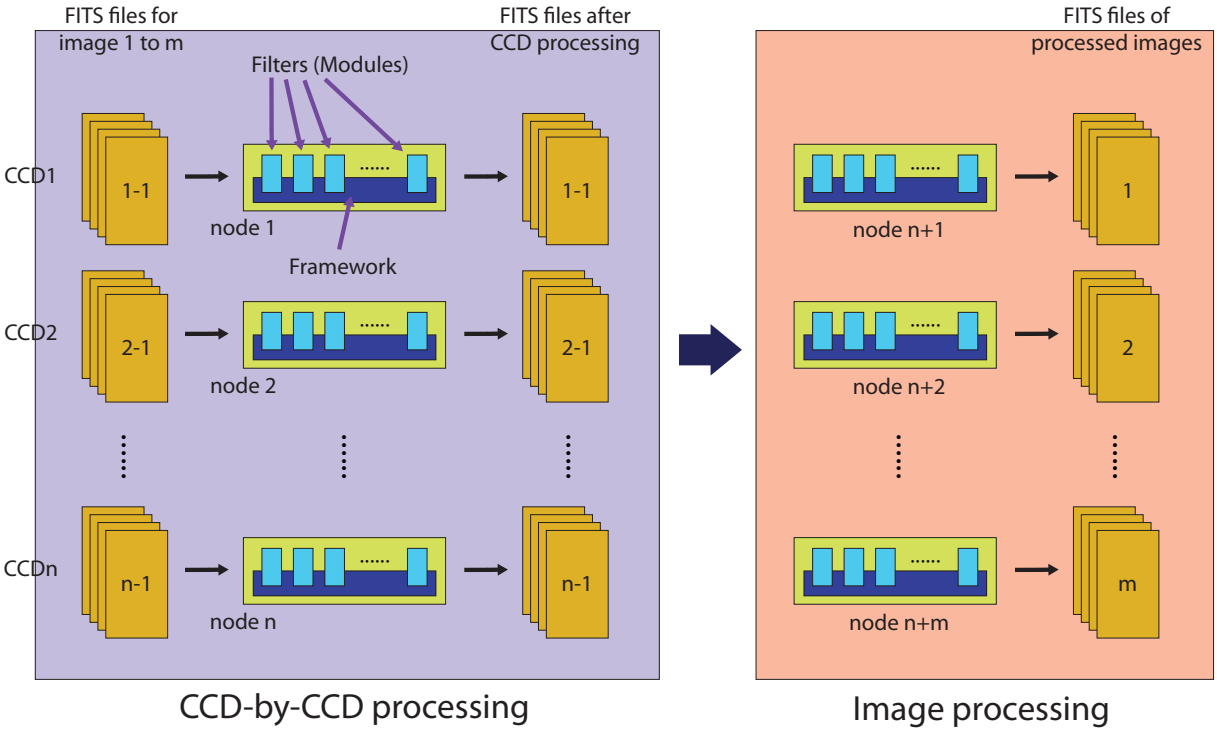


Figure 4. The real time pipelining in the HSC. Filters are same with modules. The framework is engaged in CCD-by-CCD processing and image processing.

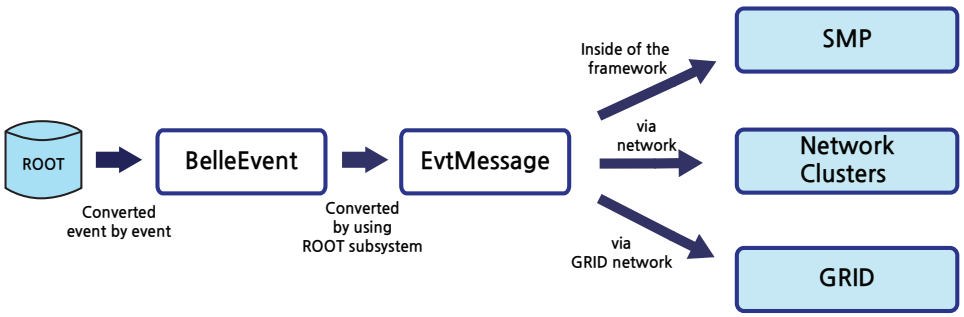


Figure 5. The object persistency inside the framework.

object. The restored BelleEvent object is used for actual data processing. After the processing, the event process converts processed BelleEvent object to the EvtMessage object, serializes the EvtMessage object, and sends it to another FIFO. Finally, the output server collects processed data from the FIFO, restores them to BelleEvent object, and reconstructs them as an output ROOT file. If the output server is in separated node, it sends the EvtMessage object to network instead of converting and reconstructing data.

5.2. Parallel Processing

The parallel processing is another key functionality of the framework. The amount of data to be processed in high energy physics has already exceeded capability of one CPU. The parallel processing enables efficient and powerful data processing. Our targets of computing resources

for parallel processing include network clusters and GRID and we are also considering to use computing clouds which becomes a trend of the latest distributed computing model [9]. One of our goals in parallel processing is providing a consistent interface to users. The framework selects computing resources dynamically so that we intend that end-user does not need to care about the resources in parallel processing.

Figure 7 shows the parallel processing scheme of the framework focused on the case of a single machine. All components in figure 7 are independent processes and they are allocated and initialized by the framework. To keep the object persistency, all data inside this scheme flow as objects. The FIFOs are used for interprocess communication (IPC). For the case of other computing resources such as network clusters and GRID, data are received from network instead of reading input files. The interface for those cases, the message passing interface (MPI) [10] and PROOF [11] are considered and we realize that MPI is more suitable for our requirements.

Figure 8 shows more details in our FIFO data structure. The shared memory which is provided by operating system is used for IPC and the semaphore [12] is used for resource synchronization. In addition, we construct a logical structure in order to use the shared memory as a FIFO. In this logical structure, several classes are defined and they makes a hierarchy. Each classes has a few number of pointers to handle the shared memory. By using these pointers, it can act like a FIFO.

6. Testing

Figure 9 shows how the performance of the framework changes as the number of event processes varies in various environments. In this testing, 2-core, 4-core, and 8-core CPU systems are used and their detail specifications are shown in table 1 and we build a module which makes full

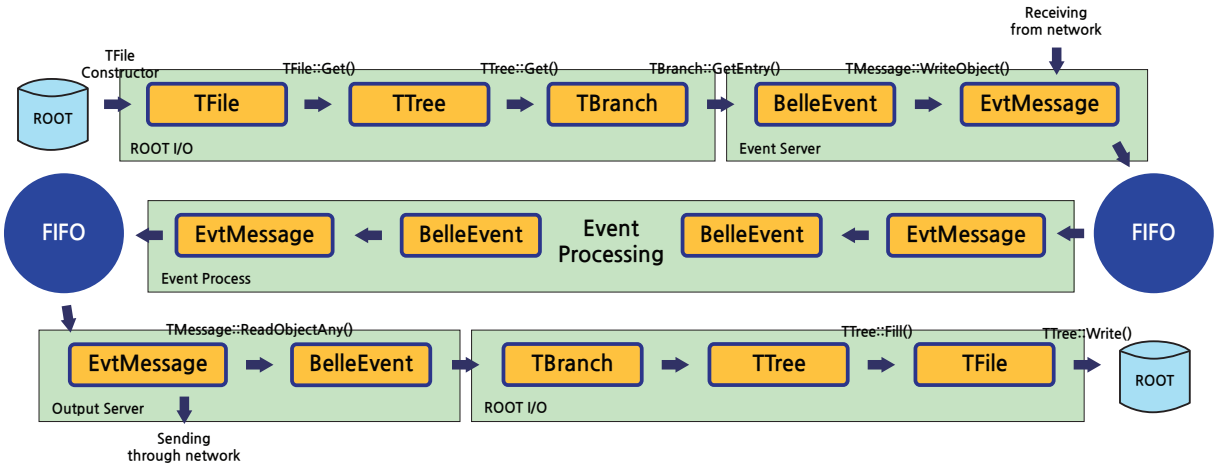


Figure 6. The object-based data flow inside the framework.

Table 1. The specifications of the testing systems.

System	CPU	Memory
2-core system	AMD Athlon64 X2 4200+ (dual core)	2 GB
4-core system	Intel Xeon 3.2 Ghz (dual core) * 2	8 GB
8-core system	Intel Xeon E5405 (2 Ghz, quad core) * 2	16 GB

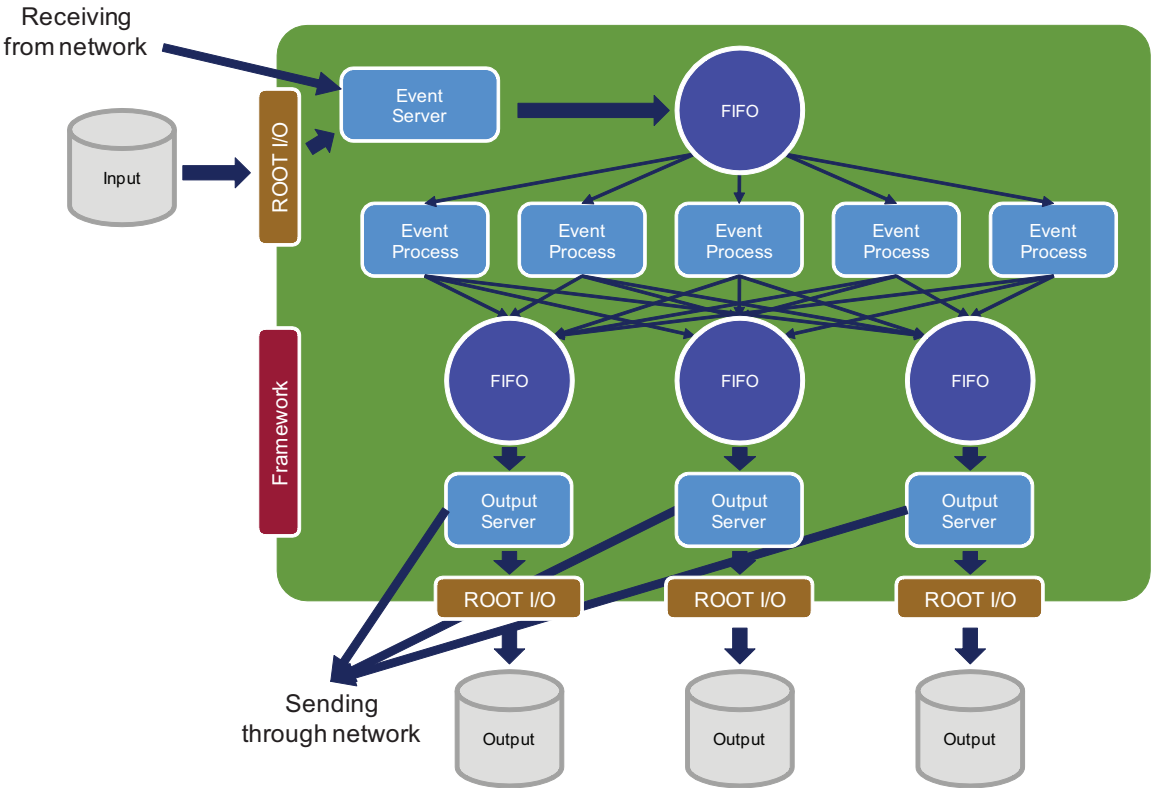


Figure 7. The parallel processing scheme of the framework for a single machine. It also illustrated for other computing resources briefly.

CPU load in order to minimize overheads from I/O. As shown in figure 9 (a), the processing time decreases gradually as the number of event processes increases. When the number of event processes reaches to the number of CPU cores of the testing environment, the processing time does not increase anymore. This is what we expected naturally. Allocated parallel event processes possess each CPU core so that the optimal performance is shown when the number of event processes becomes the same with the number of CPU cores of the testing system. Figure 9 (b) is the same plot with figure 9 (a) but vertical axis is changed to inverse of elapsed time. In this plot, we can check the linearity of performance for increasing the number of event processes.

7. Current Status

We have designed and implemented object persistency inside the framework. In order to implement it, we defined and implemented two classes of the BelleEvent and the EvtMessage class as shown in previous sections. We also have tested the implementation of object persistency. The interface for backward compatibility is also in progress. It supports conversion between current data format which is being used in the Belle experiment and ROOT data format.

In the case of parallel processing, we separated the development to two parts, module parallelization and data parallelization. The designs of both have been done. The module parallelization has been implemented and tested over network. The data parallelization has been only implemented and test inside the SMP at this moment. Now we are concentrated in implementation of the parallel processing for network clusters. The MPI is suitable for our requirement and will be used for the implementation.

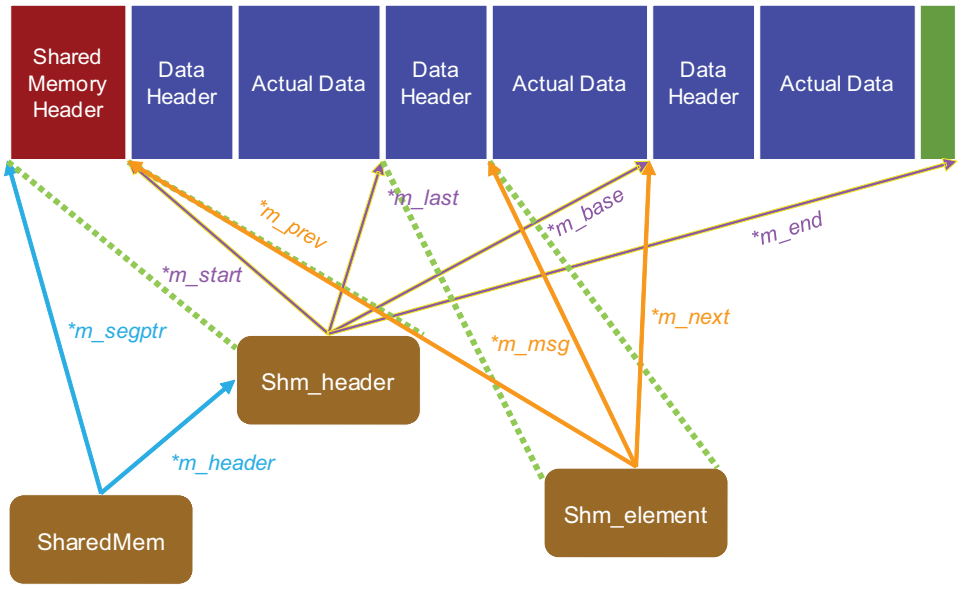
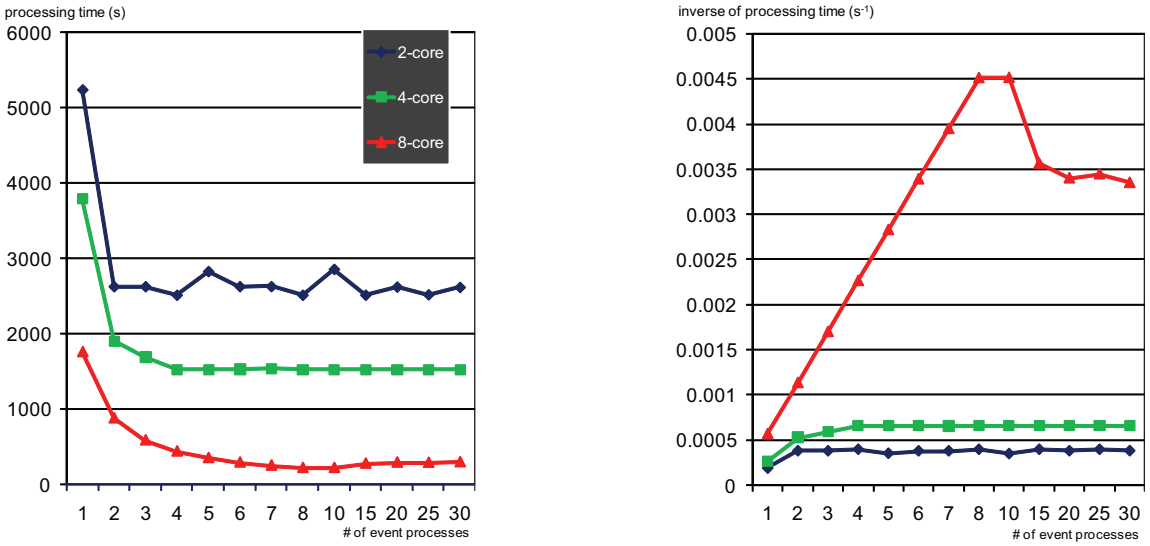


Figure 8. The logical structure of the shared memory. The SharedMem object itself is allocated at the shared memory header region and the Shm_header object itself is allocated at the data header region. The serialized data are stored at actual data region. The data is just followed after its corresponding data header. The shared memory header contains overall information of the shared memory and the data header contains information about its corresponding data.



(a) (b)

Figure 9. The performance testing result of the framework. The horizontal axis denotes the number of event processes and the vertical axis denotes the elapsed time of processing. (a) The performance behavior as a function of event processes. (b) The linearity of processing time in various number of event processes.

8. Conclusion

The upgraded experiments require high performance and flexibility of the framework since they produce much larger data than current experiments. To accomplish high performance of the framework, we are preparing interfaces for several computing resources which enable to process massive data efficiently. And we also pursuit full object persistency of the framework in order to make the framework be flexible and compatible with ROOT. The development of the framework is at the earlier stage now but we believe that our framework will provide enough flexibility and high performance for the requirements of the experiments. The fundamental structure and implementation have been developed and the implementations of the parallel processing for various computing resources such as network clusters and GRID are being developed and planned.

Acknowledgments

This work was supported by the second stage of the Brain Korea 21 Project in 2009.

References

[1] Abashian A et al. 2002 *Nucl. Instr. and Meth.* A**479** 117-232
[2] Adachi I et al. 2004 *Nucl. Instr. and Meth.* A**534** 53-8
[3] Miyazaki S et al. 2002 *Publ. Astron. Soc. Japan* **54** 833-53
[4] Abe K et al. 2004 *Letter of Intent for KEK Super B Factory* KEK Report 04-4 available from http://belle.kek.jp/superb/loi
[5] Miyazaki S et al. 2006 *Proc. of SPIE* **6269** 62690B
[6] Rademakers F and Brun R 1998 *Linux Journal* **51**
[7] Rohlfs R 2004 *ASP Conference Series* **314** 384 available from http://isdcul3.unige.ch/ rohlfs/astroroot
[8] Adachi I et al. 2003 Computing System for the Belle experiment *Preprint* physics/0306120
[9] Hayes B 2008 *Communications of the ACM* **51** 9-11
[10] Gropp W et al 1996 *Parallel Computing* **22** 728-828
[11] Ballintijn M 2006 *Nucl. Instr. and Meth.* A**559** 13-16
[12] Djikstra W E 1967 *Proc. of the 1st ACM symp. on Operating System Principles* 10.1-10.6