Lustre in WAN Environment and Development

T. Stibor, B. Alborea, S. Haller, C. Huhn, D. Klein, B. Neuburger, M. Pausch, V. Penso, C. Preuß, T. Roth, W. Schön, and J. Trautmann
GSI, Darmstadt, Germany

Lustre in WAN Environment

Recently, Lustre has been started to be employed in wide area networks (WAN) to enable efficient data access beyond local HPC and cluster environments [2]. In WAN environments challenging problems such as reliable tera-link connections and security related issues such as authentication and data privacy/integrity [1] need to be addressed. For addressing the first issue a 120 GBit/s high-speed connection between GSI and the LoeweCSC based on LNET routers was implemented (see Fig. 1). This is the first step for the general tera scale project in the context of FAIR enabling partnered research institutes Lustre access for processing data.

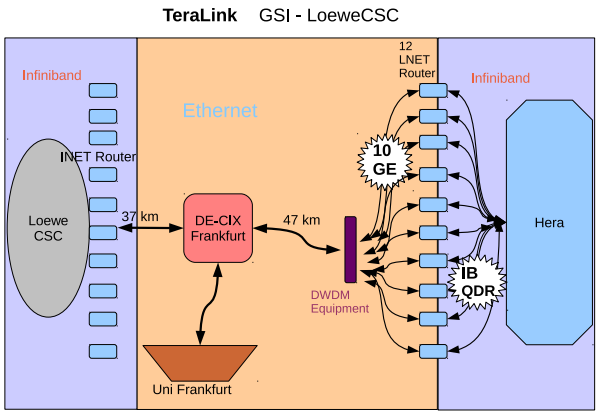

Figure 1: The high-speed connection is realized by bundling 12 machines equipped with 10 GBit/s ethernet cards acting as LNET routers. For seamless connecting both sides efficiently, Infiniband over IP is employed and network bandwidth saturation in experiments is achieved.

In the subsequent Section we briefly summarize the HPC development of a lightweight access control mechanism for addressing security related problems of Lustre in WAN environments.

A Lightweight Access Control Mechanism for Lustre in WAN Domains

For controlling access to Lustre of clients outside the GSI domain, a Linux kernel module based on Linux UID/GID and Lustre network identifier is developed. It allows to control *read* and *write* access on Lustre mount points comparable to Lustre's root squash functionality, however, for arbitrary specified UID's/GID's and Lustre network identifier ranges such as UID:$[1001 \ldots 2500]$,GID:$[2001 \ldots 3500]$, etc. In the context of wide area Lustre deployment the proposed mechanism enables a straightforward and lightweight access control of Lustre clients located in different wide area domains. The access control mechanism is implemented as a separate Linux kernel module and exports an access-granting function which is hooked into MDT system calls such `mdt_reint_open(...)` or `mdt_md_create(...)` to grant or deny access. For defining and removing rules, the Linux procfs is employed (see examples below).

Load kernel module & set rules:

```
>insmod ./lugac.ko
GSI Lustre UID/GID access control module v0.3 loaded
>echo "192.168.[67-70].[1-16]@tcp0 [500-600] 1012" > /proc/lugac
>echo "10.10.1.1@tcp5 [100-200] [100-200]" > /proc/lugac
```

Enforce rules:

```
user1@10.10.1.1:uid:5:gid:5>ls /lustre
ls:  cannot access /lustre:  Permission denied
user2@10.10.1.1:uid:105:gid:105>ls /lustre
data.raw
```

For more details and for studying the kernel module code see http://www.stibor.net/lugac.

Contributing Code to Lustre

The HPC group is now an active member of contributing and reviewing code for the Lustre file system, especially in the area of Kerberos and GSS. This encompass patches for latest Linux kernels as well as fixes for Kerberos and GSS (see http://git.whamcloud.com).

Outlook

The next project steps will encompass implementing a *kerberized* Lustre for $3.X$ Linux kernels to enable strong user authentication and data encryption/integrity. In addition, preliminary explorations and efficiency experiments on ZFS as a backbone file system for Lustre are performed.

References

[1] Josephine Palencia, Robert Budden, and Kevin Sullivan. Kerberized lustre 2.0 over the wan. In *Proceedings of the 2010 TeraGrid Conference*, pages 1–5. ACM Press, 2010.

[2] Stephen C. Simms, Gregory G. Pike, and Doug Balog. Wide area filesystem performance using lustre on the teragrid. In *Proceedings of the TeraGrid 2007 Conference*, 2007.

doi:10.15120/GR-2014-1-IT-01

Improving the logging infrastructure of HPC and Linux services

M. Dessalvi
GSI, Darmstadt, Germany

Introduction

Logging system events, especially for big IT infrastructures, is essential. Whenever a problem arise System Administrators turn their looks towards log files but as IT infrastructures grows in size and complexity the volume of the available logs will increase dramatically as well the resources needed to analyze them.

Overview

The GSI HPC group have already implemented multiple solutions, based on open source software, to analyze different kind of logs and events. A brief list of those software inlcude: Nagios, Netdisco, Collectd, MRTG graphs and Torrus.

Analyzing text logs files, produced by hundreds of hosts, is a daunting task. Traditionally, looking into such a massive amount of informations requires the use of specific utilities for searching, parsing, manipulating and extracting useful data. On UNIX/Linux systems these tools are usually available directly from the shell: cat, tail, grep, sed, awk , sort and many others. Unfortunately, these tools are not enough to spot trends and make correlations with different events scattered on multiple files.

To overcome this problem a solution was implemented, based on Logstash[1], Redis[2] and Elasticsearch[3], plus Kibana[4] as a web interface.

Logstash

Logstash is a tool for managing events and logs. It can be used to collect logs, parse them, and store them for later use.

The main idea behind this tool rely on the concept of plugins. A combination of different plugins let the admins create their own log pipeline and extract informations coming from different sources and store the results with different storage solutions.

A short list of the available plugins:

- Input: TCP/UDP, text files, Syslog, MS Windows Event logs, STDIN, etc.
- Filters: alter, collate, geoip, key value, metrics, multiline, XML, Zeromq, etc.
- Output: CSV, email , Graphite, StatsD, Elasticsearch, Nagios, XMPP, etc.

Different Logstash agents may be deployed to deal with logs from different sources and structures. The parsed results, in JSON format, is then pushed to Redis which acts as a broker between multiple Logstash agents and the Elasticsearch server.

Elasticsearch

Elasticsearch is basically a text indexing engine. It is based on Apache Lucene[5], a full-featured text search engine library written entirely in Java. Elasticsearch most interesting characteristic is its fast search responses, based on the concept of *inverted index*: instead of searching text strings directly, it creates an index from incoming text and performs searches on the index rather than the content. The Kibana web interface add an extremely flexible web interface for visualizing the collected data in real time.

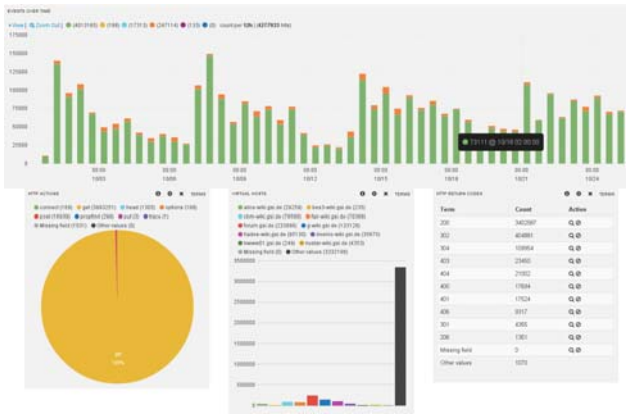

Figure 1: An example of the Kibana web interface for the Apache logs.

Outlook

Future developments of this solution will include a migration to the new 1.0 stable branch of Elasticsearch and extract even more informations from the logs through the combined use of Graphite[6] and Statsd[7].

References

[1] http://www.logstash.net
[2] http://www.redis.io
[3] http://www.elasticsearch.org/overview/elasticsearch/
[4] http://www.elasticsearch.org/overview/kibana/
[5] http://lucene.apache.org/core/
[6] http://graphite.wikidot.com/
[7] https://github.com/etsy/statsd/

 doi:10.15120/GR-2014-1-IT-02

Status of the FairRoot framework

M. Al-Turany[1], *D. Bertini*[1], *R. Karabowicz*[1], *D. Klein*[1], *D. Kresan*[1], *A. Lebedev*[1], *A. Manafov*[1], *A. Rybalchenko*[1], *T. Stockmanns*[2], *F. Uhlig*[1], *and N. Winckler*[1]

[1]GSI, Darmstadt, Germany; [2]FZJ, Juelich, Germany

Introduction

Using FairRoot one can create simulated data and/or perform analysis within the same framework. The framework delivers base classes which enable the users to construct their detectors and/or analysis tasks in a simple way [1, 2]. The user analysis is organized in tasks (FairTask). Tasks are executed sequentially in the order they are added to the run manager. One way to improve the performance of FairRoot on multi-core processors is to run the independent tasks in parallel. Which can be done by running each task in a separate thread or as separate process. However, by multi-threading an error in one thread can bring down all the threads in the process where in multi-process the different processes are insulated from each other by the OS or even the network, i.e: An error in one process cannot bring down directly another one. On the other hand, the inter-thread communication is faster than interprocess one. Trying to find the correct balance between reliability and performance we decided to use the multi-process concept with message queues for data exchange, i.e: Each "Task" is a separate process, which can be also multithreaded, and the data exchange between the different tasks is done via messages. This concept allow us to create different topologies of tasks that can be adapted to the problem itself, and the hardware capabilities. Some of the advantages of such a system are:

- Flexibility: Different data paths can be modeled by simply changing the topology of Tasks.
- Scalability: Spread the work over several processes and machines on the fly.
- Adaptive: Sub-systems are continuously under development and improvement and can be exchanged individually.

FairMQ: Base for data transport layer in FairRoot

Extending FairRoot to support multi-processes needs a solid and stable communication layer that should handle the whole communication in a transparent and stable way. Fortunately, such a layer exists already and is called ZeroMQ [3]. ZeroMQ is a very lightweight messaging system, specially designed for high throughput and low latency scenarios. It is open source, embeddable socket library that redefines the term socket as a general transport endpoint for atomic messages. ZeroMQ sockets provide efficient transport options for inter-thread, inter-process and inter-node communication (see Figure 1). Moreover it provides a Pragmatic General Multicast pattern (PGM), which is a reliable multicast protocol [3].

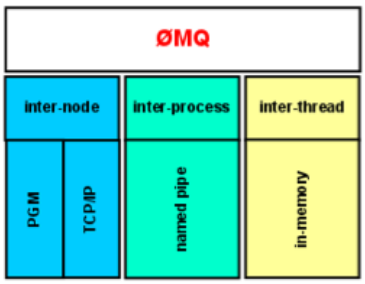

Figure 1: Message transport options in ZeroMQ. **Named Pipe**: Piece of random access memory (RAM) managed by the operating system and exposed to programs through a file descriptor and a named mount point in the file system. It behaves as a first in first out (FIFO) buffer

The zero in ZeroMQ refers to a culture of minimalism that permeates the project. They want to add power by removing complexity rather than exposing new functionality [4]. The library provides a built-in routing strategies for one-to-many or many-to-one communication scenarios. Each socket potentially comes with a sending and a receiving queue of configurable sizes.

The FairMQ package is described in more details by A. Rybalchenk in this report [5]

Outlook and future plans

We need to design and develop a dynamic deployment system (DDS) that can utilize any resource management system (e.g: Slurm, GridEngine, ...etc.). A monitoring and logging system is also under development (See report by Anar Manafov [6]).

References

[1] M. Al-Turany et al. The FairRoot framework *J. Phys.: Conf. Ser. 396 022001, 2012.*

[2] FairRoot: http://fairroot.gsi.de.

[3] http://www.zeromq.org/

[4] P. Hintjens, http://zguide.zeromq.org/page:all

[5] A. Rybalchenk and M. Al-Turany, Streaming data processing with FairMQ

[6] Anar Manfov, DDS: A New Dynamic Deployment System, this report

ALFA: A new Framework for ALICE and FAIR experiments

M. Al-Turany[1], *P. Buncic* [3], *P. Hristov*[3], *T. Kollegger* [1,2], *V. Lindenstruth*[1,2], *and P. Vande Vyvre*[3]
[1]GSI, Darmstadt, Germany; [2]FIAS, Frankfurt, Germany; [3]CERN, Geneva, Switzerland

Introduction

The FairRoot framework [1] started in 2003/2004 as a framework for CBM collaboration. Meanwhile it is used by 7 experiments as a base for their simulation and analysis: CBM, PANDA, R3B, ASYEOS and the GEM subgroup of FOPI at GSI/FAIR. The MPD experiment at JINR in Dubna and the EIC collaboration are also using the FairRoot framework as a base for their own software. The commonalities between ALICE and the FAIR experiments and their computing requirements lead to the expectation that large parts of the software framework can be written in an experiment independent way. The FairRoot project has already shown the feasibility of the approach of developing a common framework for several experiments. We therefore propose to develop the new common framework which will be called ALFA.

Technology background

The efficient use of a concurrent computing system requires the correct sequencing of the interactions or communications between different computational executions, as well as coordinating access to resources that are shared among executions. A number of different methods can be used to implement concurrent programs, such as implementing each computational execution as an operating system process, or implementing the computational processes as a set of threads within a single operating system process. The future framework has to support a heterogeneous and distributed computing system.

A message-based approach will allow us to run our software on all hardware platforms (from a laptop to an entire system with many thousands of cores and specialized hardware accelerators). An Open-source system such as ZeroMQ [2] and/or nanomsg [3] will be used as a lightweight messaging layer. What is today a single threaded application will be transformed into many small components running concurrently as independent processes (executing on the same node or distributed over the network) with some of them utilizing capabilities of specialized hardware (where available) and communicating by asynchronous messaging.

Proposed architecture

The proposed architecture will rely on a data-flow based model. It will contain a common transport layer. Common configuration, management and monitoring tools will be developed. The framework will provide unified access to configuration parameters and databases. It will include support for a heterogeneous and distributed computing system. The proposed architecture will also incorporate common data processing components.

Expected benefits from the common project

This common development will benefit to all experiments involved; it will shorten the time to deliver production quality code and will reduce the cost to develop it. The work in common will also allow a better coverage and testing of the code. Moreover, the extended user community will provide high quality documentation, training and examples.

A common framework will be beneficial for the FAIR experiments since it will be tested with real data and existing detectors before the start of the FAIR facility. For example, concepts for online calibrations and alignment can be tested in a real environment, similar to that of the planned FAIR experiments.

ALICE will benefit from the work already performed by the FairRoot team concerning features (e.g. the continuous read-out), which are part of the ongoing FairRoot development.

Outlook

A proof of concept for the design and the technology has been successfully implemented [4]. Work on different prototypes is just starting; for ALICE and CBM the prototypes are planned for the end of this year. Both prototypes should show the ability of the ALFA framework to transport huge amount of data and distribute it on a large cluster of compute nodes. For PANDA experiment a prototype is also planned for this year which will concentrate on using heterogeneous architecture (CPU and GPU) for the online event reconstruction. Implementing these prototypes simultaneously will help us identifying more common issues and will enhance the synergy between the deferent experiments. Naturally all common packages will be implemented in the ALFA framework.

References

[1] M. Al-Turany et al. The FairRoot framework *J. Phys.: Conf. Ser. 396 022001, 2012.*

[2] http://www.zeromq.org/

[3] http://nanomsg.org

[4] M. Al-Turany et al. Extending the FairRoot framework to allow for simulation and reconstruction of free streaming data . Accepted for publication by, Journal of Physics: Conference Series (2013)

 doi:10.15120/GR-2014-1-IT-04

Streaming data processing with FairMQ

A. Rybalchenko[1] *and M. Al-Turany*[1]
[1]GSI, Darmstadt, Germany

The FairRoot framework [1] is a framework for simulation, reconstruction and data analysis of particle experiments. Currently the framework is being extended to provide support for simulation, reconstruction and analysis of free streaming data. A number of components were introduced to the framework, under the common name of FairMQ, which allow flexible construction of topologies to distribute and process free streaming data. FairMQ is based on a prototype framework [3], developed in 2012/2013 at GSI, which relies on the ZeroMQ library for the transport and organization of data between system processes and/or network nodes. An example topology is presented in Figure 1. It includes two separate nodes (blue), each containing a number of processes (green and grey). Arrows indicate the data flow.

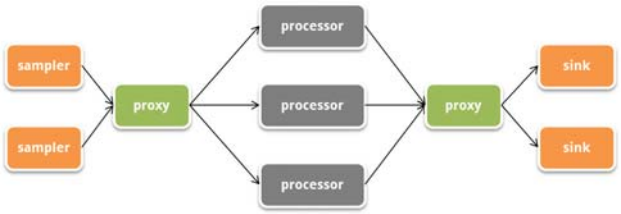

Figure 1: FairMQ example topology.

An example is implemented in FairRoot tutorials, which shows the workflow within the new scheme [7]. The example workflow demonstrates how to configure and run a topology, including components to read, process, distribute, collect and write data. The topology can be configured to run on a single node or on any number of nodes, allowing the user to monitor the system load more granularly and to add additional workers dynamically when necessary. A performance comparison with the traditional workflow was presented in [2], demonstrating significant performance benefits. On a 2 x 2.4 GHz Intel Xeon quad core node, the traditional execution setup with 8 parallel tasks reached a throughput of 2660 events/s. With the same data, FairMQ streaming system reached 7320 events/s, utilizing 9 system processes - 3 for task execution, 4 for I/O and 2 for data distribution.

Among the improvements which were introduced to FairMQ is a new component called the proxy. The proxy replaces splitter and merger components, which were used for merging/splitting of the data stream. The Proxy is more flexible than splitter/merger, because new nodes/processes can be added to it dynamically, without changing the configuration or even stopping the process. Using the proxy also simplifies the configuration, since it only occupies one network port for input and one for output, while splitter/merger would occupy a port for every new connecting node.

Moreover, FairMQ has a new layer that abstracts the specific transport implementation. By keeping framework and transport code independent, the framework can be easily adapted to emerging technologies in the future.

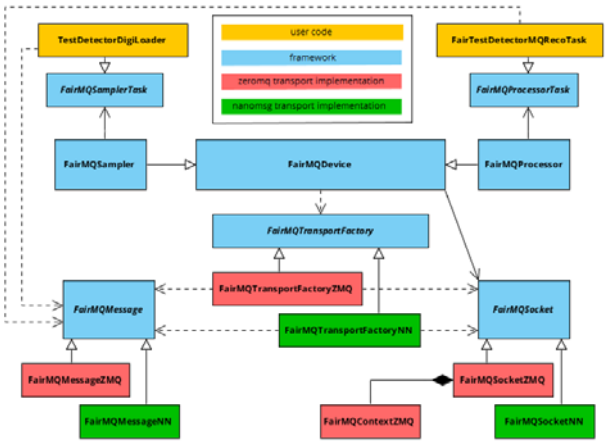

Figure 2: Transport layer abstraction.

At the moment two transport implementations can be used: ZeroMQ [4] and nanomsg [5]. Nanomsg is an emerging communication library, developed by the authors of ZeroMQ. While still in an early alpha stage, nanomsg aims to offer a number of advantages in comparison to ZeroMQ [6], which could to be useful for FairRoot. Most notable among these are better Zero-copy support (with shmem and RDMA) and a formal API for adding new transports such as Infiniband and Websocket.

References

[1] FairRoot: http://fairroot.gsi.de.

[2] M. Al-Turany, D. Klein, A. Manafov, A. Rybalchenko, F. Uhlig: Extending the FairRoot framework to allow for simulation and reconstruction of free streaming data. accepted for publication by, Journal of Physics: Conference Series (2013).

[3] D. Klein: Flexible data transport for the online analysis in a particle physics experiment. Bachelor thesis, Hochschule Darmstadt (2013).

[4] ZeroMQ: http://zeromq.org/.

[5] nanomsg: http://nanomsg.org/

[6] Differences between nanomsg and ZeroMQ: http://nanomsg.org/documentation-zeromq.html

[7] FairRoot Tutorial 3: https://github.com/FairRootGroup/FairRoot/tree/master/example/Tutorial3

DDS: A new Dynamic Distribution System

A. Manafov[1], *A. Lebedev*[1], *and M. Al-Turany*[1]
[1]GSI, Darmstadt, Germany

Dynamic Distribution System (DDS) is a tool-set that automates and dramatically simplifies the process of distribution of user defined processes with their dependencies on any resource management system using a given topology.

DDS is a successor of PoD [1][2]. Unlike PoD, which automates PROOF [3] deployment, DDS will handle any kind of user processes with complex dependencies between processes. The system is designed and being implemented within the new ALFA framework [4].

Concept

During 2013 a conceptual design of the system has been developed.

A key point of this design is the so called ẗopology languageï.e.: DDS is a user oriented system the definition of the topology by the user has to be simple and powerful at the same time. The basic building block of the system is a task. Namely, a task is a user defined executable or a shell script, which will be deployed and executed by DDS on a Resource Management System. To describe dependencies between tasks in a topology we use properties. For example, if one task wants to communicate with another task they can have the same property of a type TCP/IP port. In this case DDS will notice that, will find out a free port number on a destination system and set this number to configuration files of both user processes. Moreover, there will be different types of properties, for example, tasks can be dependent on each other via a file or a named pipe.

Tasks can be grouped into DDS collections and DDS groups. The difference between collections and groups is that collections are a signal to DDS topology parser that tasks of given collections will be executed on the same physical machine. This is useful if tasks have a lot of communication or they want to access the same shared memory. A set of tasks and task collections can be also grouped into task groups.

DDS utilizes a plug-in system in order to use different job submission front-ends. The first and the main plug-in of the system will be an SSH plug-in, which can be used to dynamically turn a bunch of machines to user worker nodes. The SSH plug-in is a perfect tool for a Cloud based solutions.

Outlook

The DDS system is in the initial development phase. We expect a lot of new features in the upcoming year.

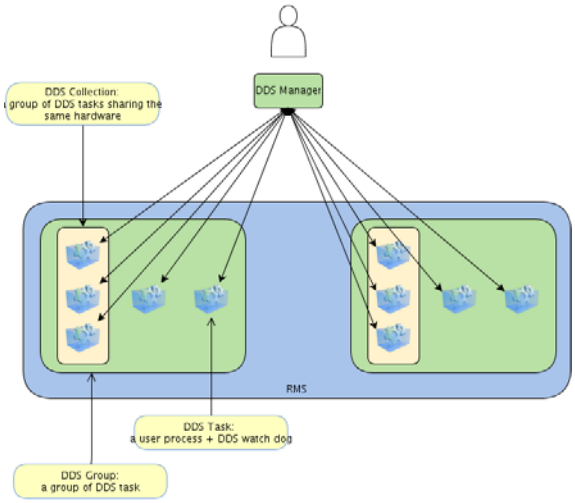

Figure 1: Illustration of the DDS concept.

1. Operational prototype of DDS with the ability to parse typologies such as examples of FairROOT tasks.
2. The operational prototype will be able to deploy single dependency tasks like a PROOF cluster or FairROOT analysis tasks. The deployment will be steered using the SSH plug-in on the Cloud and local computers.
3. Extend the plug-in machinery to cover the following Resource Management Systems: SLURM, LSF, Torque (PBS), Grid Engine, Condor, PanDA.
4. Extend the submission machinery to be able to deploy multi-dependency tasks.
5. Extend the complexity of the topology language to support custom types of task properties.

References

[1] A. Manafov et al, “PROOF on Demand”, IT-07, GSI Scientific Report 2012.

[2] PROOFonDemand(PoD),http://pod.gsi.de

[3] TheParallelROOTFacility(PROOF), http://root.cern.ch/drupal/content/proof

[4] M. Al-Turany et al., Status of the FairRoot framework, this report.

 doi:10.15120/GR-2014-1-IT-06

Event building in FairRoot

R. Karabowicz[1], *M. Al-Turany*[1], *D. Bertini*[1], *D. Klein*[1], *D. Kresan*[1], *A. Lebedev*[1], *A. Manafov*[1], *A. Rybalchenko*[1], *T. Stockmanns*[2], *F. Uhlig*[1], *and N. Winckler*[1]

[1]GSI, Darmstadt, Germany; [2]FZJ, Juelich, Germany

Introduction

This report presents first effort of constructing a common structure of event building in the FairRoot [1] computing framework. Set of classes has been implemented allowing management and development of event builders working on different data streams. A working example of event reconstruction using data from one of the PANDA detectors, the GEM Tracker, has been provided. The achievement of the full event reconstruction depends on the implementation of other event builders working on different data sets. We anticipate that the provided structure will also be applicable to other experiments facing similar challenges.

Event building

In general, the event building requires information from most of experimental subsystems. Some will provide good event start time, others are designed to reconstruct particle trajectories, and yet another serve for particle identification. The complexity of the task suggests usage of different event builders for separate subsystems and then combining the information in an event builder manager to get a global picture of the event.

It should also be noted, that the event builders might have radically different functionalities. It is easy to imagine, that some of the subsystems will be able to provide crucial event characteristics and thus will be used for event reconstruction. However some will only be able to assign data to already identified events and thus will merely build up events. Trivially many will serve both goals.

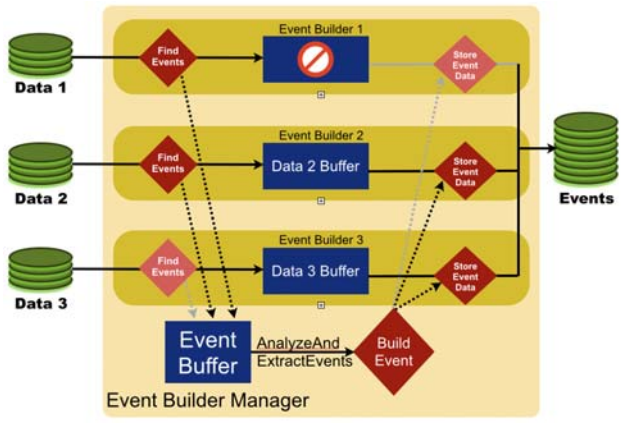

Figure 1: Schematic view of event building.

Figure 1 schematically presents a prototype design of the event building scenario. The reconstruction is handled by the Event Builder Manager, which contains 3 (in this example) independent event builders, that are getting time-sliced data from different sources. Event builders process input data in *FindEvents* functions, which may store the data in internal Data Buffers and/or send found events information to the Event Builder Manager.

The task of combining the information from different subsystems is performed by the *AnalyzeAndExtractEvents* function of the manager, which, in turn, triggers storing of the data for each identified event. This is performed by the *StoreEventData* function of the event builders, where the data in buffers have to be assigned to events.

Example implementation

GEM Tracker event builder was the first implementation of the presented scenario. The input for the event builder are the time slices with reconstructed particle trajectories in the GEM Tracker [2]. For each track an estimated track creation time is calculated using GEM timing information. Event builder looks for tracks with similar (closer than 5ns) creation time and calculates event time by taking center-of-gravity average. Even single trajectories are taken to mark reconstructed events due to the small detector average occupancy of around 3 tracks per event.

Around 80% of the realistic antiproton-proton collisions have reconstructable trajectories in the GEM Tracker. Out of them more than 90% have been properly reconstructed using the presented analysis scenario. About 2% of the events in the output have no matching simulated event.

Summary

This report addresses the question of the event building in the future experiments at FAIR. It proposes a common structure for such tasks within the FairRoot framework. Data from the different subsystems would be analyzed by different event builders and the whole effort would be coordinated by the experiment-specific Event Builder Manager. A preliminary example of the event builder operating on the data from only one of the PANDA detectors shows promising results. The future work should focus on the development and improvement of the event builders for different experimental subsystems as well as on the Event Builder Managers, that are foreseen to orchestrate the various event builders and take the final decision in the process of the event reconstruction.

References

[1] M. Al-Turany *et al*, "Status of the FairRoot framework", GSI Scientific Report 2013

[2] R. Karabowicz, "Time-based reconstruction in the GEM Tracker", GSI Scientific Report 2013

doi:10.15120/GR-2014-1-IT-07

New software for the R^{3}B calorimeter CALIFA within FairRoot*

Y. González[1,2] *and D. Kresan*[1]
[1]GSI, Darmstadt, Germany; [2]USC, Santiago de Compostela, Spain

Introduction

R^{3}B is a next generation experimental setup for studies of Reactions with Relativistic Radioactive Beams. Its aim is to provide a versatile reaction setup for kinematically complete measurements of reactions with high-energy radioactive beams [1]. CALIFA, the R^{3}B CALorimeter for In-Flight emitted pArticles detection, is one of the key detectors for the setup of R^{3}B.

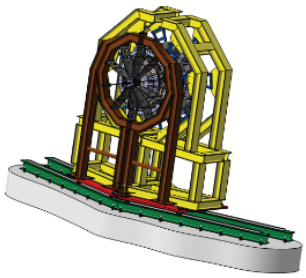

Figure 1: CALIFA calorimeter in last year report [2].

As CALIFA is included in the R^{3}B experiment, the software requirements will be handled by the FairRoot framework.

CALIFARoot

The framework FairRoot provides base classes which enable the users to construct their detectors and tasks in a straight-forward way [3]. Two important parts of the software for CALIFA have been developed for the framework: the unpacker and the raw-data publisher. Both of them will be fully explained in the next sections.

CALIFA unpacker

The unpacker is the element that allows us to extract the information obtained via the data aquisition system and save it in ROOT format. In order to create the unpacker for CALIFA we need three essential elements: a representation of a hit in one of the crystals, the unpack code to fully read the information obtained and to extract it, and a macro that provides information regarding how the unpacker will be handled and creates the ROOT file with the resulting data and histograms.

CALIFA raw-data publisher

The raw publisher, fully based on a javascript code called JSRootIO [4], is an useful feature for experiments that involve different institutes, as it allows them to start live analysis while the experiment is still running.

In order to do this, a webpage is generated and has to be uploaded to a server, along the ROOT file where the data shown is taken from. Two elements are necessary: the generator of the web file and code to add the histograms to the ROOT file that can be read with the javascript code. The webpage is currently able to use the javascript code both via an external link and with a local deployment, downloading the required files in the server.

Conclusion

Basic tools for CALIFA have been developed during the year 2013. Extended software, like database infrastructure, is expected to be developed during the year 2014.

References

[1] R^{3}B collaboration, `http://www.gsi.de/R3B`

[2] R^{3}B collaboration, "Status of the CALIFA/R^{3}B calorimeter", GSI Scientific Report 2012 (2013) 198

[3] FairRoot, `http://fairroot.gsi.de/`

[4] JSRootIO, `http://root.cern.ch/js/`

* Work supported by Ministerio de Economía y Competitividad(Spain) contract No. EIC-FAIR-2011-0061

 doi:10.15120/GR-2014-1-IT-08

Remote Event Client Implementation in FairRoot Framework

D. Kresan, F. Uhlig, and the FairRoot Group
GSI, Darmstadt, Germany

Introduction

FairRoot is a framework, which is used by future FAIR experiments for an implementation of the simulation, reconstruction and data analysis algorithms [1]. Requirement of both, the basic framework features and the user-defined tasks developments, is the usage of the same code for online and offline event reconstruction. In this report we will present an extensions to several base classes, which was done to provide the possibility for communication with a Data Acquisition (DAQ) server.

Online Data Processing

The possibility of online event reconstruction is an essential point for most of the experiments at FAIR, which are aiming to perform high-intensity measurements without a hardware trigger. Therefore, we implemented an additional functionality to the framework, which is sketched in figure 1. In order to avoid overloading of the main DAQ

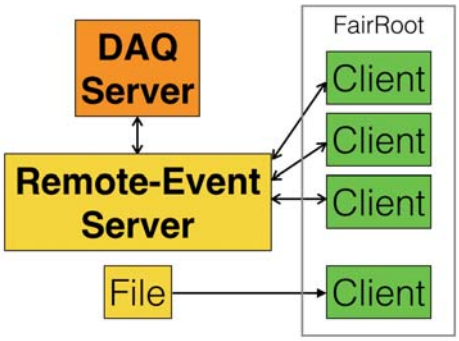

Figure 1: Illustration of the communication flow between a DAQ server and the FairRoot framework.

server with connections from multiple clients, an intermediate stage called "Remote-Event Server" [2] was introduced. There is also a possibility to connect a client directly to a file on disk, in order to support also the offline analysis. The class design allows supporting multiple data formats from a server or from a saved file. In the next subsection we shortly describe the Multi Branch System (MBS) [3] format used in many DAQ server implementations at GSI.

MBS Format

The data coming from the DAQ contains response from multiple detectors and is formed in sequences of events, each containing several sub-events. Each sub-event corresponds to one detector. A detector can be then identified via TYPE and SUB_TYPE values in the sub-event header. This header in a sub-event is then followed by the data.
A support for other data formats is provided, as was already requested by the CBM collaboration.

Server Side

The Remote-Event Server is to be started on a separate computer node. It connects directly to the specified DAQ server, and transmits the data using ROOT sockets to a specified network port. The name of the node, where this server is started, has to be used in a client configuration, described in the next section. For the testing purpose, one can send events with random numbers without connecting to a DAQ server.

Client Side

The Remote-Event Client is part of FairRoot. The steering class *FairRunOnline* implements an event-loop, and user-specified tasks execution. It gets the data from an object of the concrete implementation of the abstract base class *FairSource*. Detector-specific "unpacker" classes then process the received raw data events. The unpackers fill the data as arrays of ROOT objects, which are taken as input for the hardware calibration and further analysis.
This dedicated class design allows a user to easily switch in his analysis between online-streaming and stored data by changing just one line in the steering macro.
Further developments will focus on a class *FairRootSource*, which will be inherited from *FairSource* and will read simulated data from a ROOT file. This will allow to combine two steering classes *FairRunAna* (currently used for the analysis of Monte Carlo simulations) and *FairRunOnline* into a single implementation.

Summary

We presented in this report an extension to the FairRoot framework for communication with a Remote-Event Server. The implementation will be used in the detector tests for the future R^3B experiment. These experimental tests provide unique possibility for validation and verification of the dedicated detector-specific reconstruction algorithms.

References

[1] http://fairroot.gsi.de
[2] H. Göringer, http://www-aix.gsi.de/∼goeri/mbsnew/online.html
[3] H.G. Essel, N. Kurz, http://web-docs.gsi.de/∼mbs/v51/manual/gm_mbs_c.pdf

doi:10.15120/GR-2014-1-IT-09

FairDB: The FairRoot Virtual Database

Denis Bertini[1]
[1]Scientific Computing ,GSI, Darmstadt, Germany

Introduction

Since the setup of a typical heavy ion experiment at FAIR [1] is varying depending on the physic case to study, the FairRoot framework for simulation and analysis [2] should be able to handle a large amount of different parameters and their variation in time. Currently the FairRoot framework uses the Hades Runtime Database Library [3] as third-party code to initialize parameter from ascii or ROOT [4] input files. This report describes the new FairDB Virtual Database Library which extends the FairRoot framework initialization scheme to any SQL-Databases systems.

Database Interface

FairDB provides the user with a well-defined and uniform API for database manipulation. Using internally the ROOT `TSQLServer` class [4], the same user code can be executed independently on Oracle, MySQL, PostGresSQL and even SQLite SQL database engines (Fig. [1]).

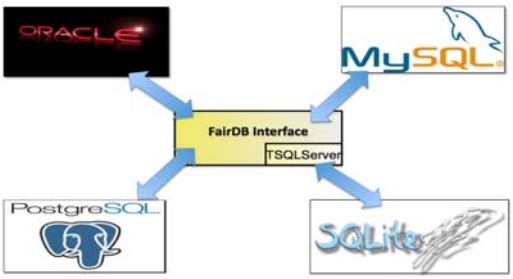

Figure 1: FairDB supported SQL-Database engines.

Design

FairDB separates the connectivity to the DBMS into a *front-end* and a *back-end* (Fig [2]). The *front-end* delivers the relational data model and implementation (tables, query algorithm, versioning etc ...) together with the database connectivity layer. The *back-end* defines the communication layer to specific DBMS using the appropriate client library. Recently the PostGresSQL lower level driver `TPgSQLServer.cxx` has been corrected and partially rewritten. The patch has been accepted and is part of the ROOT release 5-34-15.

Initialization

The SQL-Database input can be added to the FairRoot initialization scheme (Fig [3]) using the services of the general parameter input `FairRunTimeDB`. For accessing the data more than one database connection can be used according to a given user-defined URLs list.

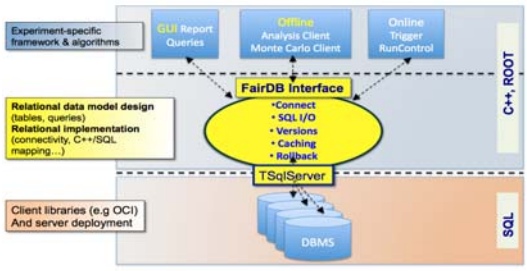

Figure 2: FairDB Design

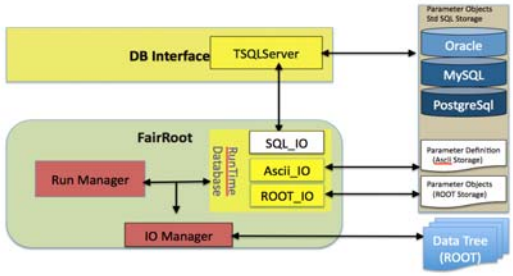

Figure 3: FairRoot initialization scheme including SQL Database input.

User Manual

In order to smooth the learning curve for the FairDB usage, a complete User Manual in PDF format has been written. It covers the general concept, database installation and all the FairDB interface features with tutorials [5].

References

[1] http://www.gsi.de/fair/

[2] The FAIR simulation and analysis framework 2008 J. Phys.: Conf. Ser. 119 032011

[3] http://web-docs.gsi.de/ ilse/initialization.htm

[4] R. Brun, F. Rademakers, P. Canal, I.Antcheva , D. Buskulic, O. Couet, A. and M. Gheata *ROOT User Guide* CERN, Geneva 2005.

[5] FairRoot Virtual Database (User Manual). `https://panda-wiki.gsi.de/foswiki/pub/Computing/PandaRoot/FairRootVirtualDatabase.pdf`

 doi:10.15120/GR-2014-1-IT-10

Time Based Version Management with FairDB

Denis Bertini[1]
[1]Scientific Computing ,GSI, Darmstadt, Germany

Introduction

Because of unprecedented interaction rate (10^7/s) and data rates from 100Gb/s up to 1Tb/s, experiments at Fair [1][2] employ self-triggered front-end electronics featuring a continuously sampling system where the detector subsystems are synchronized by a precision time stamp distribution system. In this context of free data streaming, time or more precisely time interval is the natural detector buffer identifier.
For this reason, FairDB [3] implements internally a version management for the parameter fully based on time.

Temporal Database

Conceptually in a relational database data is organized in two-dimensional space (attribute, tuple). FairDB implements a time-based version management by adding the time as a third dimension leading to a three dimensional model (attribute, tuple, time) as shown in Figure [1].

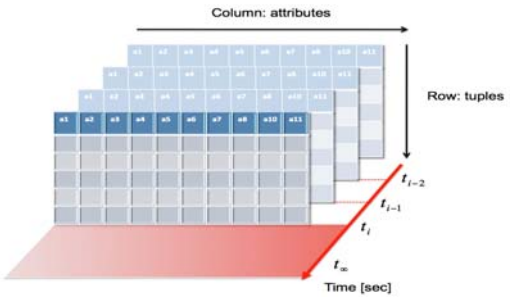

Figure 1: FairDB timed based versioning uses time as additional dimension to the relational data model.

Relational Model

FairDB separates the metadata table (logical data) from the payload data table (physical data). It uses a two-level hierarchy to store the physical data which is illustrated in Figure [2]. This two-level hierarchy data access corresponds to a two-table structure for each type of parameter data in the FairDB relational model. The two table are connected via a unique identifier primary key used as a sequence number.

Validity Range

Ultimately, any of the data retrieved could depend on the run or the time interval defining the event being processed. Detector relevant parameters, such as calibration constants,

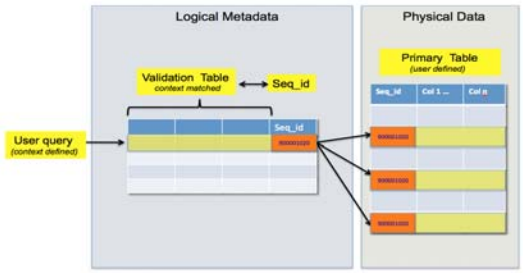

Figure 2: Data Table location Hierarchy

will change with time and the interface has to retrieve the correct ones for the current event time-slice. For this reason, all requests for data through the interface must supply information about:

- The date and time interval of event (in UTC)
- The type of data: real or Monte Carlo
- The type of Detector.

In FairDB this information is called a Context. In the database all information is tagged by a Context i.e by a validation range which identifies the type of data and detector and the ranges of date times for which it is valid.

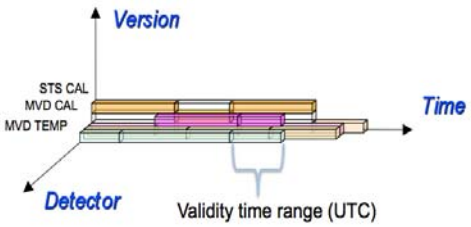

Figure 3: FairDB Multidimensional version management. Only the Validity Time Range is mandatory for parameter retrieval.

References

[1] The Data Acquisition and Trigger System of Panda, 2008 IEEE N57-1

[2] Simulation and reconstruction of free-streaming data in CBM, 2011 J. Phys.: Conf. Ser. 331 032008

[3] FairRoot Virtual Database (User Manual). `https://panda-wiki.gsi.de/foswiki/pub/Computing/PandaRoot/FairRootVirtualDatabase.pdf`

doi:10.15120/GR-2014-1-IT-11

FairDB SQL Persistency Scheme

Denis Bertini[1]
[1]Scientific Computing ,GSI, Darmstadt, Germany

Introduction

Conceptually FairDB [1] handles data as table i.e in a two-dimensional data structure with cells organized in rows and columns.
This is the reponsability of the FairDB framework to deliver an automatic parameter object to relational table mapping and a uniform API to access these object regardless of the internal relational representation.

Object to Table Mapping

FairDB eases the transition parameter object to the relational data model by implementing an automatic mapping procedure according to the following rules:

- **Class** definition corresponds to table definition
- **Columns** are the physical equivalent of the attributes
- **Rows** are the physical equivalent of objects instances

Additionnaly each record (rows or unit block of rows) on a table is uniquely identified. The purpose of the unique identifier is to act as the *primary key* on the table where it is defined and to be referenced as the *foreign key* by other related tables.
Figure [1] shows an example of object to table mapping in the case where the target parameter class is a simple class or is related through inheritance.

Figure 1: One-to-One Parameter object to relational table mapping.

SQL I/O scheme

FairDB provides a programmatic and uniform template based API to write and access parameter object. Like the Java SQL interface system (JDBC)[2], each data retrieval produces a pointer giving read access to a results table which corresponds to a subset of the underlying database table (Figure [2]). Each Row of the results table corresponds to a object, the type of which is user defined and table-specific.

Minimizing I/O with Level2 Cache

Another important point is to minimise I/O. Some requests, particularly for detector relevant parameters, can pull in large amounts of data but users must not load it once at the start of the job and then use it repeatedly since it may not be valid for all the data they process. Also multiple users may want access to the same data and it would be inefficient for each to have their own copy.
To deal with both of the above problems, the interface uses the concept of handle or proxy . When accessing a particular table, a table-specific pointer object is created. the corresponding object is usually very small is suitable to be stack based and passed by value, thus reducing considerably the risk of a memory leak.
During construction of the pointer, a request for data is passed down through the interface and the results table, which could be large, is created on the heap. The interface places the table in its cache and the user's pointer is attached to the table, but the table is owned by the interface, not the user.

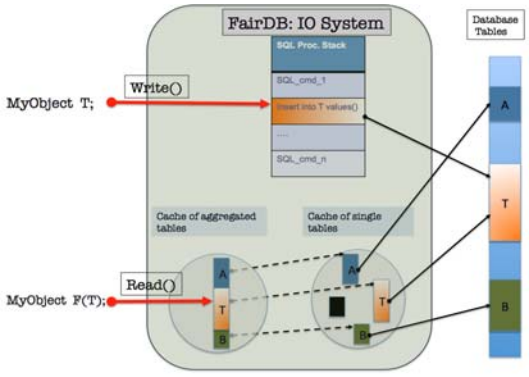

Figure 2: FairDB SQL I/O persistency scheme. Different level of Caches are used to store recently accessed table in on disk.

References

[1] FairRoot Virtual Database (User Manual). `https://panda-wiki.gsi.de/foswiki/pub/Computing/PandaRoot/FairRootVirtualDatabase.pdf`
[2] http://docs.oracle.com/javase/tutorial/jdbc/

 doi:10.15120/GR-2014-1-IT-12

E-Science Activities at GSI

K. Schwarz for the e-science group/Scientific Computing[1]
[1]GSI, Darmstadt, Germany

This article describes the work of the GSI e-science Group with the aim to operate an ALICE tier2 centre within the global environment of the LHC Computing Grid and to prototype a distributed computing environment for FAIR.

ALICE tier2 centre at GSI and ALICE Grid in Germany

The ALICE tier2 centre and the National Analysis Facility at GSI provide a computing infrastructure for ALICE Grid and for the local usage of the German ALICE groups. New data-sets are being transferred to GSI continuously and are processed on the local batch farm via daily running analysis trains. Disk space is provided via the Cluster File System Lustre. The storage resources pledged to the global ALICE community (550 TB) are provided via a Grid Storage Element which consists of an xrootd daemon running on top of the Lustre file system. Data stored on the Grid Storage Element can also be read from local ALICE jobs. The remaining Lustre capacity (4.7 PB) is used for local storage but has to be shared with other experiment groups of GSI/FAIR. Throughout the year GSI participates in centrally managed ALICE Grid productions and data analysis activities, but also analysis jobs of individual users are running on the ALICE tier2 centre. Since recently the routing problem between the GSI batch farm and the local ALICE Storage Element has been solved the average job efficiency of Grid jobs running at the ALICE tier2 centre is close to that of other high level tier 2 centres. The overall job share in 2013 contributed by GSI tier2 and Forschungszentrum Karlsruhe (ALICE tier1 centre) as well as the HHLR compute cluster at Goethe University in Frankfurt has been 11% of all ALICE Grid jobs worldwide. This corresponds well with the pledged CPU resources for 2013: 7000 HEP-SPEC06 for GSI tier2 (4% of the global T2 requirements) and 30000 HEP-SPEC06 for FZK (30% of the global T1 requirements)

CRISP and LSDMA

The Cluster of Research Infrastructures for Synergies in Physics (CRISP) project is a collaboration between different institutions and facilities related to physics research. GSI participated in Work Package 16 and contributed in developing a pan-European system for unique identification. As a prototype solution Umbrella has been presented. The work of this group concentrated on bridging solutions between the Umbrella system and X509 certificates as used in Grid communities as well as between Umbrella and EduGAIN.

The work of the Data Life Cycle Lab ”Structure of Matter/FAIR” within the portfolio project ”Large Scale Data Management and Analysis (LSDMA)” is being defined in close collaboration with the FAIR experiments. Main work topics are parallel and distributed computing as well as global data federations.

KOSI

Within the context of the KOSI program (Kooperativer Studiengang Informatik) from Hochschule Darmstadt there is a computing project in close collaboration with the GSI theory group. It explores the synergetic use of different software packages like Mathematica and MathCode from Wolfram with Geneva from Gemfony Scientific and with Lapack, ScaLapack and GSL on the GSI computer clusters. A first application on the chiral extrapolation of baryon masses has been worked out [1].

Preparation for FAIR

In order to be able to prepare a distributed computing concept for FAIR experience is being accumulated starting with currently existing computing environments. Currently the focus lies on the AliEn middleware since know-how is available due to the ALICE tier2 centre at GSI. Based on a gap analysis it will be decided if the FAIR computing requirements can be mapped on further developed current systems or if major components if not all have to be replaced. Well-grounded decisions can be done by evaluating the performance of test beds. Therefore PANDA-Grid came into existence and is now in successful operation since 2004. It has been extended to currently 15 sites in Asia, Europe, and North America. Among these 3 sites are running via WLCG infrastructure. The most recent site which joined is Orsay in France and also the GSI Icarus cluster has been interfaced to PandaGrid. The good collaboration between ALICE Offline and the PandaGrid project is being continued, so recently a common workshop in Torino took place. The central PandaGrid services as well as the central Grid data base (MySQL) are running at GSI while the central MonaLisa monitoring repository and a backup database are maintained in Torino.

References

[1] M.F.M. Lutz, K. Schwarz, R. Bavontaweepanya, and C. Kobdaj, “On finite volume effects in the chiral extrapolation of baryon masses”, this report.

doi:10.15120/GR-2014-1-IT-13

High-Level Data Flow Description of FPGA Firmware Components for Online Data Preprocessing*

H. Engel[1], *F. Grüll*[1], *and U. Kebschull*[1]

[1]IRI, Institut für Informatik, Johann Wolfgang Goethe-Universität Frankfurt, Frankfurt, Germany

FPGA firmware for detector read-out is commonly described with VHDL or Verilog. Data processing on the algorithmic level is a complex task in these languages and creates code that is hard to maintain. There are high level description frameworks available that simplify the implementation of processing algorithms. A sample implementation of an existing algorithm and the comparison with its VHDL equivalent show promising results for future online preprocessing systems.

Field Programmable Gate Arrays (FPGAs) are widely used in high energy physics detector read-out chains due to their flexibility. The protocols and interfaces are usually implemented with hardware description languages like VHDL or Verilog. With FPGAs getting bigger and faster they become more and more suitable for performing complex data processing tasks. This can reduce the data volume and significantly ease demands on later software based processing steps. The drawback of the commonly used hardware description languages is that they are mostly working on the Register Transfer Level. This is perfect for high performance protocol and low level interface implementations. However, using these languages to implement data processing on an algorithmic level requires experienced developers and usually involves customized IP cores and latency matching of components. This creates a rather complex and static design. There are several high level hardware description frameworks available that provide their own languages to describe data processing steps on an algorithmic or data flow level. Some of them also come with an own framework including building blocks for PCIe or DRAM interfaces. This significantly speeds up the development compared to a description in plain VHDL or Verilog.

The underlying framework of this work is made by Maxeler Technologies. The platform generates a pipelined version of the algorithm after its data flow graph has been described in a Java-like programming language [1]. The compiler manages the scheduling of the design, inserts latencies in the generated pipelines wherever needed to keep the data in sync, and instantiates interfaces to PCIe or DRAM if required. A software environment with a device driver and C API provides easy to use stream interfaces to the hardware. The compiler translates the data flow description into VHDL code which is then run through the vendor tools.

The algorithm described in this way is the FastClusterFinder that was used as a VHDL core in the readout of the ALICE Time Projection Chamber during LHC run period 1 [2]. A simplified overview of the algorithm is shown in Fig. 1. The incoming raw data is decoded into a data stream with time and location information. The center of gravity and the deviation of peaks are calculated in time direction. In a second step neighboring cluster candidates are merged to get the center of gravity and the deviation of the full cluster in pad direction. The last step is a floating point division. The VHDL implementation is a rather complex design due to its data flow control structures and the number of fixed point and floating point arithmetics.

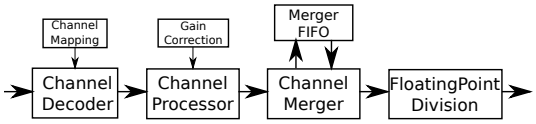

Figure 1: Schematic picture of the ALICE TPC FastClusterFinder algorithm.

A functionally identical version of the FastClusterFinder algorithm has been described with the Maxeler data flow description language. The behavior of this implementation has been verified in simulation using recorded detector data from ALICE TPC. The output of the data flow simulation is directly compared to the output of a Modelsim simulation of the original VHDL code. In comparison to the VHDL implementation the number of lines of code is significantly reduced for the data flow description. Especially the computing intensive parts of the design are very easy to understand and to maintain. The resource usage of the generated design in its current state is slightly different in details but overall in the same order of magnitude as the VHDL implementation.

This implementation shows that there are tools available to describe processing algorithms on an algorithmic or data flow level that are able to generate hardware with comparable resource usage but significantly reduced code volume. This greatly improves maintainability of the code. A next step will be to implement and test the code in actual hardware. Furthermore, the generation of VHDL code out of the data flow description allows the processing elements to be extracted from the vendor framework and integrated as IP core into an own firmware environment.

References

[1] Maxeler Technologies, *Programming MPC Systems*, White Paper, June 2013

[2] T. Alt and V. Lindenstruth, *Status of the HLT-RORC and the Fast Cluster Finder*, GSI Scientific Report 2009

* Work supported by HGS-HIRe, HIC4FAIR

 doi:10.15120/GR-2014-1-IT-14

gStore – the GSI Archive Storage for Experiment Data

H. Göringer, M. Feyerabend, M. Imhof, and S. Sedykh
GSI, Darmstadt, Germany

Overview

gStore is a client/server middleware developed at GSI and tailored according to the requirements of the GSI experiments. gStore provides high performance access 24 hours a day and 7 days a week. For running experiments, highly parallel online writing to dedicated gStore write cache is enabled, including online data copy to lustre for online experiment monitoring and analysis. Due to the design principles:

- reliable long term archive storage,
- full scalability in data capacity and I/O bandwidth,
- high performance access due to intrinsic parallelism,

gStore is well prepared for the challenges of the future FAIR T0 centre. Design principles and functionality are described in detail in GSI reports, talks, and two papers.[1]

Hardware Status

Experiment data are archived in two automatic tape libraries (ATLs), which are also used for backup of user data. The larger ATL has a storage capacity of 8.8 PByte and an I/O bandwidth of 2 GByte/s currently. The smaller ATL (1.3 PByte) is located in the remote BG2 building and contains copies of experiment (raw) and user backup data. This concept prevents from loss of valuable data in case of media damage and enables data recovery even in case of a disaster in the computing centre.

Data are accessed via data movers with read and write disk cache of 220 TByte overall. This large disk buffer hides the tape storage from the users to a big extent. The lustre file system /hera, a cluster file system with ˜ 3 PByte storage capacity, is the online storage for data analysis on Prometheus. The I/O bandwidth between gStore and /hera amounts to 2.5 GByte/s currently.

gStore Enhancements

Data copy to lustre. To utilize the available bandwidth, for retrieve processes from tape to lustre automatical parallelization by gStore has been implemented. The input data are sorted twice, according to their location on tape media and to the storage order on the media. Then files on different tape media are copied in parallel by different processes running on different data movers, and all files are read in optimal order from the corresponding media. Obviously the number of parallel processes is limited by the number of available tape drives, which is up to eight currently. It should be noted that this parallelization cannot be done by the users themselves, because they intentionally need not care about file location on tape and therefore do not have the corresponding information. Due to the high performance connection between gStore and lustre - matching the tape speed of 250 MByte/s - tape files are copied directly to lustre, skipping the otherwise mandatory staging step to gStore read cache. This additionally reduces the copy time considerably.

The parallelization concept works only efficiently if a large number of files is copied with one single (gstore) command, the more files are involved, the better. To specify many files, wildcard characters, a file list, recursive file operations, or any combination of them can be used. Thereby users need not keep track of files already archived in gStore or already staged or retrieved from gStore, respectively, because files already existing are never overwritten, except if explicitly specified otherwise. If some of the specified gStore files are not on tape, but already staged in read cache or still residing on write cache, additional copy processes on each data mover involved are started. Therefore a few tens of copy processes may initiated by one single user command

Removal of limitations in file number. To support as many files as possible in single commands, some limitations in file number have been identified and removed. Up to now with one single command more than 100,000 files have been processed successfully by users.

As an example, in a recursive user archive command 127,473 matching files have been found in lustre. 26,635 files have been rejected, mainly because they were already existing in gStore, or because they were empty or had invalid file names. The remaining 100,838 files, with file sizes from some 10 bytes to ˜ 1 GByte and ˜ 6.5 TByte size in total, have been archived successfully to the write cache of a data mover. With all latencies included, the average data rate amounted to 152 MByte/s.

Outlook

It is planned to implement automatic parallelization also for processes copying from lustre to gStore. Using up to 10 data movers in parallel, the overall copy time would be decreased by an order of magnitude.

Newer lustre versions soon available at GSI enable to implement and test the lustre HSM (Hierarchical Storage Manager) functionality with tape backend in gStore.

According to the road maps of big tape manufacturers, data capacity (now 4 TByte/tape) and I/O speed (now 250 MByte/s) will be doubled in the next years. With additional frames for tape media and tape drives, our ATL data capacity could be enhanced then to ˜ 100 PByte, which is already the order of magnitude needed for FAIR.

References

[1] see `http://www.gsi.de/informationen/wti/it/exp_daten/daten_speicherung_e.html` as starting point for more info

doi:10.15120/GR-2014-1-IT-15

 doi:10.15120/GR-2014-1-IT-15