Efficient Time Frame Building for Online Data Reconstruction in ALICE Experiment

Efficient Time Frame Building for Online Data Reconstruction in ALICE Experiment

A. Rybalchenko[1], M. Al-Turany[1],[2], C. Kouzinopoulos[2], N. Winckler[1] for the ALICE Collaboration

[1]GSI Helmholtzzentrum fr Schwerionenforschung GmbH, Planckstrae 1, 64291 Darmstadt, Germany

[2]CERN, European Laboratory for Particle Physics. 1211 Geneve 23, Switzerland

E-mail: a.rybalchenko@gsi.de

Abstract.
After the Long Shutdown 2 period, the upgraded ALICE detector at the LHC will produce more than a terabyte of data per second. The data, constituted from a continuous un-triggered stream data, have to be distributed from about 250 First Level Processor nodes (FLPs) to about 1500 Event Processing Nodes (EPNs). Each FLP receives a small subset of the detector data that is chopped in sub-time frames. One EPN needs all the fragments from the 250 FLPs to build a full time frame. An algorithm has been implemented on the FLPs with the aim of optimizing the usage of the network connecting the FLPs and EPNs. The algorithm minimizes contention when several FLPs are sending to the same EPN. An adequate traffic shaping is implemented by delaying the sending time of each FLP by a unique offset. The payloads are stored in a buffer large enough to accommodate the delay provoked by the maximum number of FLPs. As the buffers are queued for sending, the FLPs can operate with the highest efficiency. Using the time information embedded in the data any further FLP synchronization can be avoided. Moreover, zero-copy and multipart messages of ZeroMQ are used to create full time frames on the EPNs without the overhead of copying the payloads. The concept and the performance measurement of the implementation on a reference computing cluster are presented.

1. Introduction

The ALICE experiment is focused on analyzing lead-ion collisions using proton-proton, proton-lead and lead-lead collisions as references [1]. During the Long Shutdown phase in 2018/19, the ALICE detector at CERN will be upgraded and re-launched. The physics objectives after the upgrade require high statistics due to low cross sections and very small signal-to-background ratio. After the upgrade, the classic trigger system cannot be used and the main detectors (TPC, ITS) will operate in continuous mode. The throughput from the continuous data stream from the detector is expected to be greater than 1 TB/s. The output data stream is fed into approximately 250 First Level Processor nodes (FLP). The data from these streams will be analyzed online, reduced by a factor of 2.5, merged, chopped into manageable pieces called time frames and sent to the approximately 1500 Event Processing Nodes (EPN), where it will be further reduced to meet the storage requirements [2]. One EPN requires sub-time frames from all FLP nodes in order to build a full time frame and forward it to the global processing instance.

The pattern of the data transfers between FLPs and EPNs dictates very high requirements for the underlying network. Fig. 1 provides a high level overview of the network that connects the FLP units to the EPN processing layer. It must be capable to sustain a high throughput traffic (2.5-5 Tbit/s) to assemble the time frames data from each FLP into one EPN computing unit. All of the sub-time frames that become available at the frame dispatching step at the same time have to be transferred to a single EPN node. Combined with the size of a full time frame of 11 GB [2], this requires every link in the network chain to be able to sustain high throughput and contention. Several network technologies are under investigation for the final systen [2]. The technology used for the final system will be decided upon at a later stage, to make use of new technology developments and potential cost benefits. The transport system must take this into account.

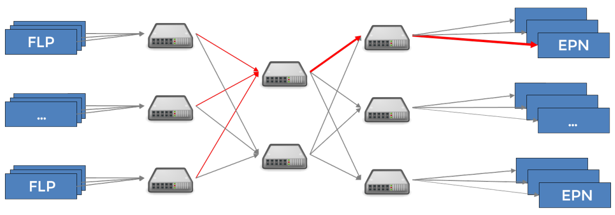


Figure 1. High level overview of the network architecture. Approximately 250 FLP (First Level Processor) nodes distribute the data to about 1500 EPNs (Event Processing Node) over a switching network. The highlighted paths indicate the sending pattern when all FLPs send to the same EPN, which results in very high link contention.

In this work, the time frame dispatch and building components needed by the FLP and EPN layers are designed and implemented using the FairRoot/FairMQ software layer [3, 4], within the ALICE O^2 (Online-Offline) computing framework [2]. The components meet the high throughput requirements of the system. They are shown to utilize the full available bandwidth on the test nodes connected by 40 Gbit/s Ethernet or QDR (Quad Data Rate) (40 Gbit/s) InfiniBand networks. The scalability of the components to a large number of nodes is demonstrated. To avoid blocking, all time-consuming operations are implemented in an asynchronous way. To help reach maximum performance, unnecessary copying of the data is avoided. The FLP components are aware of the EPN availability and react to its changes by preventing the data being queued to unavailable EPNs. Traffic shaping has been implemented on the FLPs to minimize contention when several FLPs are sending sub-time frames to the same EPN. The application of the traffic shaping results in a balanced resources usage and a predictable traffic pattern. To abstract the traffic shaping from the underlying network technology, it is implemented on the application level. This makes it possible to switch to a different network technology in the future.

Using FairMQ for the data transport provides a flexible and efficient communication tool and allows the network components to remain independent of the used network technology. It is one module of the FairRoot framework, allowing the user to run processing components that interact via an asynchronous messaging system. FairMQ offers an abstract transport interface, that is currently implemented via two communication libraries - ZeroMQ [5] and nanomsg [6], offering transport via network, inter-process and inter-thread communication. It gives the user access to several communication patterns, such as Publish-Subscribe, Push-Pull and Request-Reply, allowing a flexible design of the transport system.

2. Components for Frame Dispatch and Time Frame Building

An overview of the system design is shown in Fig. 2. The following sub-sections describe the functionality of the components in detail. The system operates in two modes - regular mode (default) and test mode. In test mode the FLP components generate the data. In regular mode the data is received by the FLP components from the local processing step on the FLP node and handled in zero-copy fashion to avoid copying.

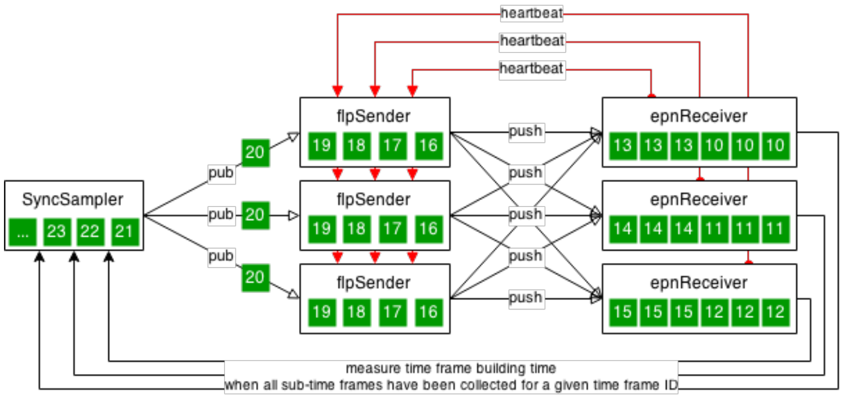


Figure 2. Example topology consisting of one SyncSampler component, three FLP components and 3 EPN components. The green boxes represent sub-time frames with their IDs. Data with the same ID needs to be delivered to the same EPNs and combined for further processing. The heartbeats provide availability information of the EPNs. A feedback channel from EPNs to the SyncSampler component allows to measure the total time frame building time.

2.1. First Level Processors

The FLP components on different nodes receive sub-time frames belonging to the same time frame roughly simultaneously. To achieve this in the test environment, an additional component is used - SyncSampler. SyncSympler also receives feedback from EPNs upon building a full time frame to measure the performance of the system. All sub-time frames that belong to the same time frame have to be routed to the same EPN component. The routing decision for the time frames is done locally in each FLP component to avoid additional synchronization that would increase the complexity of the system. Given that the algorithms in the global processing step depend on having collected sub-time frames from all FLP nodes for the reconstruction, a failure of only one FLP node would be critical to the functioning of the entire processing pipeline. Because the FLP nodes are bound to parts of the detector by physical links, their failures would cause the experiment to stop. Such failures have to be monitored and reacted upon by the global experiment control system. It is equally important to not block the FLP nodes waiting for the EPNs in order to send out the data because this can cause the memory on the FLP node to fill up rapidly in case of a slow response from EPN node.

In the test mode, each FLP component generates dummy data of configurable size upon receiving the time frame ID from the SyncSampler. The final outgoing message is a multi-part message. It's structure is presented in table 1. The multi-part structure allows efficient parsing of the header on the receiver side without touching the message content.

header (time frame ID)	data of configurable size

Table 1. Payload structure for the test mode

The FLP component maintains availability information for the EPNs via a system of heartbeats. Each EPN component publishes a heartbeat at a configurable rate. The routing decision described below takes into account the time of the last received heartbeat from a given EPN. If the last heartbeat is older than the configured time the data for it is discarded. It will be send to again when and if it sends a heartbeat again. The decision to remove the EPN completely from the list of available EPNs has to be taken centrally by the Experiment Control System [2].

The routing of the sub-time frames in the FLPs is done locally. The approach makes use of the continuously incrementing time frame ID, either retrieved from the data in the case of the regular mode, or received from the SyncSampler in the case of the test mode. Because all the sub-time frames of a frame have an identical ID, the decision on the recipient EPN for a given sub-time frame can be done locally on each FLP. The decision is done via the following formula:

$$DestinationEPN = timeFrameID \bmod numberOfAvailableEPNs$$

Based on this formula, the sub-time frames are distributed to the available EPNs on a round-robin basis, ensuring that all sub-time frames with a specific ID arrive to the same EPN.

2.2. Event Processing Nodes

The EPN components must assemble the full time frames efficiently before forwarding them to the next processing step with minimal delay. Moreover, a mechanism is needed for discarding incomplete time frames after a configurable timeout. Compared to a failure of an FLP node, a failure of an EPN node would only result in small data loss. Such a failure can be detected and as a result the node will be excluded from the list of available EPNs.

Because the order of the arriving sub-time frames is not guaranteed, the incoming fragments are handled asynchronously and stored in a separate buffer. To avoid expensive copying operations on the data, only pointers to arrived data fragments are stored with the associated time frame IDs.

The component checks the buffer periodically for incomplete time frames. Any incomplete time frames that have been present in the buffer for longer than a configurable amount of time, are discarded to avoid consuming system memory indefinitely if one or more of the sub-time frames is no longer available. The EPN component stores the discarded time frame IDs in a separate list. Should a sub-time frame appear with an ID that has already been discarded, it will not be stored, because it can never be completed.

When fully assembled, the time frames are sent out for further processing. In test mode, the data is simply discarded and only a confirmation containing the time frame ID is sent to the SyncSampler component to measure the time frame building time. In regular mode, the data is forwarded to the global processing instance.

The EPN component publishes its availability in configurable time intervals. To do this efficiently, the process is launched in a separate thread. The process does not share any state with the main functionality of the EPN component. This allows to avoid any locking mechanisms.

3. Traffic Shaping

With some network technologies, the described traffic pattern has been known to cause a serious throughput collapse problem, called TCP Incast [7]. TCP Incast occurs when the number of simultaneous data senders increases past the ability of the network switch to buffer it. As a result, packet losses may occur. When the number of lost packets reaches a certain level, the sender will issue a TCP retransmission time out (RTO), that can last at least 200 ms. The Incast problem has been extensively researched due to the increasing number of occurrence scenarios in big data applications, resulting in many solution proposals on different levels.

Although the problem of TCP Incast does not occur with InfiniBand transport due to its collision avoidance mechanisms [8], the proposed mitigation solutions for TCP can serve as an example for a more general approach of traffic shaping and contention minimization to optimize the network usage. Several solutions on the network level have been proposed, modifying various network parameters and algorithms [9, 10, 11, 12, 13]. These approaches, however, are bound to a particular network technology and require low level modifications of the network stack. To remain open to future network technologies and be able to reduce network requirements at the same time, an application-level solution is desirable. The application can have detailed knowledge about the required traffic pattern and the overall system structure, that is not available with network level solutions. Several application-level solutions to TCP Incast have been proposed by Krevat, et al. [14] and Yang, et al. [15]. Two notable solutions include staggering and global scheduling of the data transfers. The global transfer scheduling introduces an additional component to the system which increases its complexity, adds potential point(s) of failure to the system and consumes additional traffic (even though it may be minimal). Solving the problem locally with transfers staggering can avoid all of these drawbacks.

The idea behind staggering the data transfers is to avoid transmitting the data at the same time by delaying the sending on each instance by a unique offset, while keeping the data in a buffer large enough to accommodate the delay provoked by the maximum number of involved senders. A similar approach has been presented in [15], demonstrating very good results, depending on the chosen time delay value for the staggering transfers. The approach in [15] shows that the best delay values are those close to the transmission time (latency) of one data unit (message).

The staggering approach was chosen to enable traffic shaping in the frame dispatch scenario and to remain independent of the network technology. An outline of the approach is presented in Fig. 3. Because each subsequent time frame has a different EPN node as a destination, the communication can continue with maximum throughput when the staggering pattern is established and the designated buffers are full.

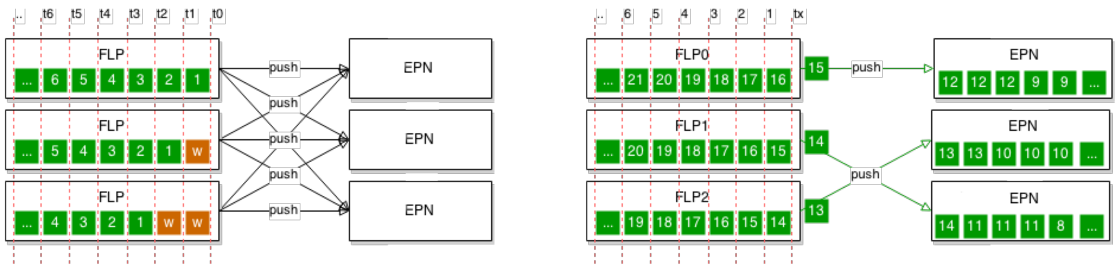


Figure 3. Example scenario of staggering transfers with priority values increasing by 1 for each FLP component (no simultaneous transfers). In this scenario, the first FLP component does not have any delay and can constantly send at full rate. Others delay the incoming sub-time frames until the path to the target EPN becomes free (the delay is shown in the left picture with 'w'). The picture on the right shows the sending pattern when the buffers for the staggering are full - each component can send at full speed and avoid collisions because the destination EPN is different for each sender.

The time delay for an outstanding sub-time frame X is calculated as follows:

$$delay = MeasuredLatencyFor(sizeof(X)) * Priority$$

The time delay can be further adjusted, according to the performance measurements of the final system. It allows any traffic pattern, depending on the configuration per FLP component and the data size. In the test mode, the start value for the delay calculation is set upon receiving the data/signal. In the regular mode, the time stamp embedded in the data can be used instead. This ensures that the FLPs will calculate the time delays based on the same information, without additional synchronization. The required additional buffer space on the FLPs depends on the desired transfer pattern. If no simultaneous transfers are desired, one of the FLP components would have to have a buffer large enough to hold a full time frame. If a number of simultaneous transfers are allowed, this requirement can be relaxed.

4. Scalability and Traffic Shaping Tests

This section discusses the way the performance of the components is affected by the number of computing nodes and the usage of the presented traffic shaping.

4.1. Test Environment and setup

The scalability of the components and the effects of traffic shaping have been tested on a computing cluster consisting of 40 nodes. The cluster consisted of nodes with two different CPU architectures. 12 dual-socket node with two Intel Xeon E5520 CPUs with 4 cores each (plus Hyper-Threading) and 24-48 GB RAM and 28 nodes equipped with dual-socket boards with two AMD Opteron 6172 CPUs with 12 cores each and 64 GB RAM. The nodes were running a Fedora Release 20 operating system with a 3.17.4 Linux kernel and were connected with a QDR (40 Gbit/s) InfiniBand network.

Because the ZeroMQ transport implementation of FairMQ transfers the data using the IP-over-InfiniBand (IPoIB) protocol, the performance of the transport also depends on the CPU architecture. The components have been shown to sustain a throughput of 2.5 TB/s per node in the demonstrator tests of the ALICE O^2 facility design [2]. Initial tests have shown that more than one process per node is necessary to reach the maximum throughput of IPoIB. For the presented scenario, the most efficient configuration was to use 2 sender processes per FLP node and 3 receiver processes per EPN node.

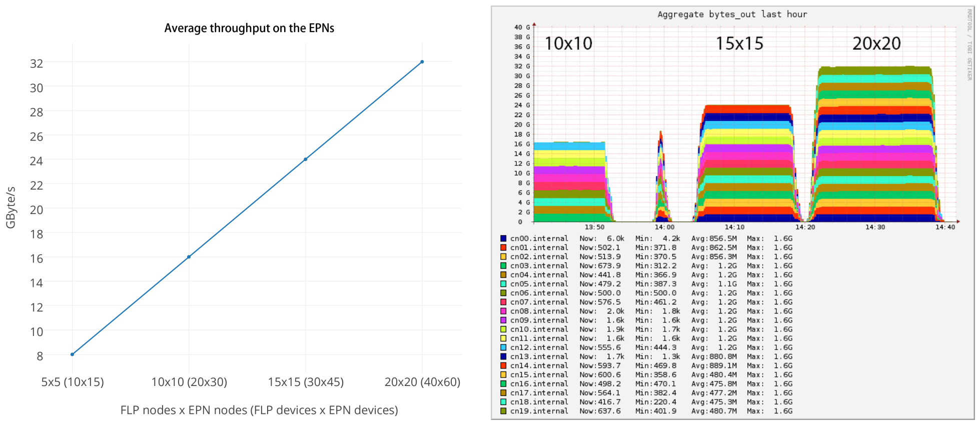
4.2. Scalability

The scalability test involved an increasing number of FLP and EPN nodes. Starting with a setup of 5x5 nodes, the numbers were increased to 20x20 nodes. The FLP components on each sender node were configured to send out sub-time frames with a constant rate of 1.6 GB/s. The results are presented in Fig. 4, showing the throughput graph and the output from the monitoring software on the nodes. No performance penalties were observed when increasing the number of involved nodes.

4.3. Traffic Shaping

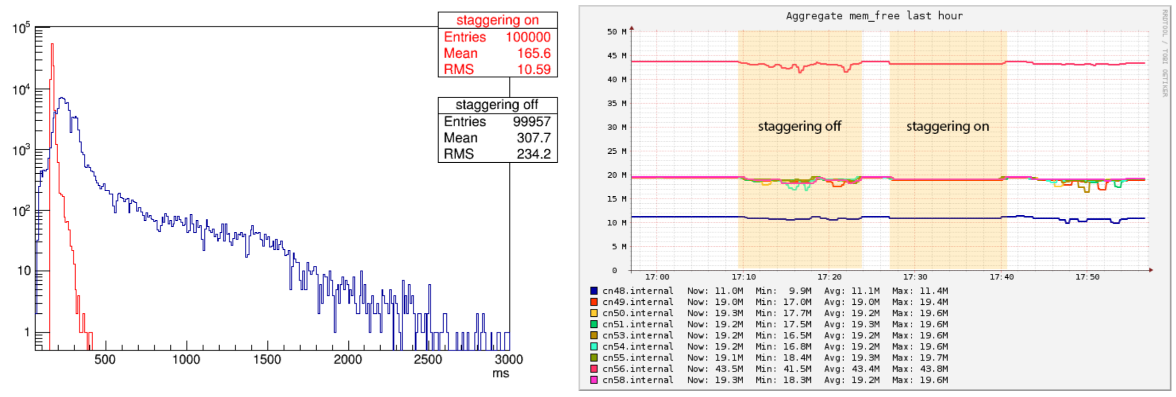
The traffic shaping test compares the time of the system to assemble a full time frame, both with and without the staggering of the transfers. For this test, 100000 time frames were transferred, each with a size of 153MB, from 18 FLP components (9 nodes) to 27 EPN components (9 nodes). Sub-timeframe size for each FLP sender is $153/18 = 8.5$ MB. The results are presented in Fig. 5, showing a histogram of the distribution of time frame building times and the monitoring output of the memory consumption on the EPN nodes. The timeframe building time includes sending the start signal from SyncSampler, generating and sending the data on the FLPs, receiving and merging on the EPNs and sending the confirmation back to the SyncSampler. Both with and without the traffic shaping, the average throughput per node was at the configured 1.6 GB/s on InfiniBand.

The results show that the mean time frame building time was reduced almost by half when applying the staggering technique for traffic shaping (from 307.7 ms to 165.6 ms). The root



```
■ cn00.internal  Now:  6.0k  Min:  4.2k  Avg:856.5M  Max:  1.6G
■ cn01.internal  Now:502.1   Min:371.8   Avg:862.5M  Max:  1.6G
■ cn02.internal  Now:513.9   Min:370.5   Avg:856.3M  Max:  1.6G
■ cn03.internal  Now:673.9   Min:312.2   Avg:  1.2G  Max:  1.6G
■ cn04.internal  Now:441.8   Min:366.9   Avg:  1.2G  Max:  1.6G
■ cn05.internal  Now:479.2   Min:387.3   Avg:  1.1G  Max:  1.6G
■ cn06.internal  Now:500.0   Min:500.0   Avg:  1.2G  Max:  1.6G
■ cn07.internal  Now:576.5   Min:461.2   Avg:  1.2G  Max:  1.6G
■ cn08.internal  Now:  2.0k  Min:  1.8k  Avg:  1.2G  Max:  1.6G
■ cn09.internal  Now:  1.6k  Min:  1.6k  Avg:  1.2G  Max:  1.6G
■ cn10.internal  Now:  1.9k  Min:  1.7k  Avg:  1.2G  Max:  1.6G
■ cn11.internal  Now:  1.6k  Min:  1.6k  Avg:  1.2G  Max:  1.6G
■ cn12.internal  Now:555.6   Min:444.3   Avg:  1.2G  Max:  1.6G
■ cn13.internal  Now:  1.7k  Min:  1.3k  Avg:880.8M  Max:  1.6G
■ cn14.internal  Now:593.7   Min:469.8   Avg:889.1M  Max:  1.6G
■ cn15.internal  Now:600.6   Min:358.6   Avg:480.4M  Max:  1.6G
■ cn16.internal  Now:498.2   Min:470.1   Avg:475.8M  Max:  1.6G
■ cn17.internal  Now:564.1   Min:382.4   Avg:477.2M  Max:  1.6G
■ cn18.internal  Now:416.7   Min:220.4   Avg:475.3M  Max:  1.6G
■ cn19.internal  Now:637.6   Min:401.9   Avg:480.7M  Max:  1.6G
```

Figure 4. Performance results when increasing the number of sender and receiver nodes/processes. Configured transfer rate per node - 1.6 GB/s. Two sender processes per node, three receiver processes per node. No noticeable penalties when increasing the number of involved components.



```
■ cn48.internal  Now: 11.0M  Min:  9.9M  Avg: 11.1M  Max: 11.4M
■ cn49.internal  Now: 19.0M  Min: 17.0M  Avg: 19.0M  Max: 19.4M
■ cn50.internal  Now: 19.3M  Min: 17.7M  Avg: 19.2M  Max: 19.6M
■ cn51.internal  Now: 19.2M  Min: 17.5M  Avg: 19.3M  Max: 19.6M
■ cn53.internal  Now: 19.2M  Min: 16.5M  Avg: 19.2M  Max: 19.6M
■ cn54.internal  Now: 19.2M  Min: 16.8M  Avg: 19.2M  Max: 19.6M
■ cn55.internal  Now: 19.1M  Min: 18.4M  Avg: 19.3M  Max: 19.7M
■ cn56.internal  Now: 43.5M  Min: 41.5M  Avg: 43.4M  Max: 43.8M
■ cn58.internal  Now: 19.3M  Min: 18.3M  Avg: 19.2M  Max: 19.6M
```

Figure 5. Effects of the traffic shaping with a setup of 18 FLP processes and 27 EPN processes. On the left: distribution of full time frame building times with 100000 time frames, 153 MB each (18 sub-time frames). One the right: Memory consumption by the EPN processes (the graph shows the amount of free memory per node).

mean square of the distribution was also significantly lower with the traffic shaping - 10.59 ms, compared to 234.2 ms (without traffic shaping). The produced balanced and predictable traffic pattern compensated for the introduced delays on the FLPs. Moreover, assembling full time frames earlier allowed them to be kept in memory for shorter periods of time (for both building and further processing). For this test only the building time was significant, since there was no actual processing of the dummy data. With additional processing time and the increased number of FLP nodes in real scenario, the memory usage will also be increased. Therefore the benefit of reducing the building time will become more important.

5. Future Work

Using IP-over-InfiniBand transfers allows for the utilization of existing communication libraries that are based on TCP/IP. However, this means that some CPU time is consumed by the protocol. Although the demonstrated performance is sufficient for the time frame dispatch scenario, it can be further increased by using native InfiniBand transfers with RDMA (Remote Direct Memory Access). Native InfiniBand support can be added to one of FairMQ transports - ZeroMQ or nanomsg, or by directly implementing the FairMQ transport interface.

The time frame building functionality in the EPN component is specific to ALICE data format. Another possible development of the component would be to allow it to handle any type of incoming data. The description of the data structure and the merging strategy would then be provided by the user, leaving the rest of the component unchanged. One possible way to implement this is by using a Policy-based design, that separates the implementation of the necessary functionality into policy classes. It allows compile-time assembly of the final component via a template system, which eliminates the dependency of the component on the implementations and performs checking of several error types at compilation time.

6. Conclusion

The presented components for the time frame building on the FLP and EPN nodes meet the high throughput requirements for the upcoming upgrade of the ALICE experiment. The scalability of these components has been demonstrated on a computing cluster equipped with QDR InfiniBand. Furthermore, a traffic shaping has been implemented on the application level on the FLP nodes to reduce network contention when several FLPs transfer data to the same EPN. The application of the traffic shaping delivers a predictable flow of the data through the network links and a more balanced usage of the available resources.

References

[1] K. Aamodt, et al.: *The ALICE experiment at the CERN LHC*, 2008, JINST, Vol. 3, p. S08002.
[2] P. Buncic, M. Krzewicki, P. Vande Vyvre, et al.: *Technical Design Report for the Upgrade of the Online-Offline Computing System*, 2015, CERN, Geneva, The LHC experiments Committee.
[3] M. Al-Turany, D. Bertini, R. Karabowicz, D. Kresan, P. Malzacher, T. Stockmanns, F. Uhlig: *The FairRoot framework*, 2012, J. Phys.: Conf. Ser. 396 022001.
[4] 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*, 2014, J. Phys.: Conf. Ser. 513 022001.
[5] P. Hintjens: *ZeroMQ: Messaging for Many Applications*, 2013, O'Reilly Media, isbn: 9781449334444
[6] M. Sustrik: *nanomsg*, http://nanomsg.org/ (accessed 2015-03-01).
[7] Y. Chen, R. Griffit, D. Zats, R. H. Katz.: *Understanding TCP Incast and Its Implications for Big Data Workloads*, 2012, EECS Department, University of California, Berkeley, UCB/EECS-2012-40.
[8] R. Noronha: *Designing High-performance and Scalable Clustered Network Attached Storage with Infiniband*, 2008, Ohio State university, AAI3326242.
[9] Y. Chen, R. Griffith, J. Liu, R. H. Katz, A. D. Joseph: *Understanding TCP Incast Throughput Collapse in Datacenter Networks*, 2009, Proceedings of the 1st ACM Workshop on Research on Enterprise Networking, 1592693, pp. 73–82, New York, NY, USA.
[10] H. Wu, Z. Feng, C. Guo, Y. Zhang: *ICTCP: Incast Congestion Control for TCP in Data Center Networks*, 2010, Proceedings of the 6th International Conference Co-NEXT '10, 1921186, pp. 13:1–13:12, New York, NY, USA.
[11] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan: *DCTCP: Efficient Packet Transport for the Commoditized Data Center*, 2010, ACM SIGCOMM.
[12] IEEE Standard *IEEE 802.1Qau Standard - Congestion Notification*, 2010, http://www.ieee802.org/1/pages/802.1au.html
[13] M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshmikantha, P. Rong, B. Prabhakar, M. Seaman: *Data center transport mechanisms: Congestion control theory and IEEE standardization*, 2008, 46th Annual Allerton Conference on Communication, Control, and Computing, pp. 1270–1277.
[14] E. Krevat, V. Vasudevan, A. Phanishayee, D. G. Andersen, G. R. Ganger, G. A. Gibson, S. Seshan: *On Application-level Approaches to Avoiding TCP Throughput Collapse in Cluster-based Storage Systems*, 2007, Proceedings of the 2Nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing '07, 1374598, pp. 1–4.
[15] Y. Yukai, H. Abe, K. Baba, S. Shimojo: *Staggered flows: An application layer's way to avoid incast problem*, 2012, Cloud Computing Congress (APCloudCC), 2012 IEEE Asia Pacific, pp. 64–67.