Studies on load metric and communication for a load balancing algorithm in a distributed data acquisition system

M Simon\textsuperscript{1,2}, H Sakulin\textsuperscript{2} and S Kozielski\textsuperscript{1}

\textsuperscript{1}Faculty of Automatic Control, Electronics and Computer Science, Silesian University of Technology, Akademicka 16, 44-100 Gliwice, Poland
\textsuperscript{2}CERN, CH-1211 Genève 23, Switzerland

E-mail:michal.simon@cern.ch

Abstract. The proposed method is designed for a data acquisition system acquiring data from \( n \) independent sources. The data sources are supposed to produce fragments that together constitute some logical wholeness. These fragments are produced with the same frequency and in the same sequence. The discussed algorithm aims to balance the data dynamically between \( m \) logically autonomous processing units (consisting of computing nodes) in case of variation in their processing power which could be caused by some faults like failing computing nodes, or broken network connections.

As a case study we consider the Data Acquisition System of the Compact Muon Solenoid Experiment at CERN’s new Large Hadron Collider. The system acquires data from about 500 sources and combines them into full events. Each data source is expected to deliver event fragments of an average size of 2 kB with 100 kHz frequency.

In this paper we present the results of applying proposed load metric and load communication pattern. Moreover, we discuss their impact on the algorithm’s overall efficiency and scalability, as well as on fault tolerance of the whole system. We also propose a general concept of an algorithm that allows for choosing the destination processing unit in all source nodes asynchronously and asserts that all fragments of same logical data always go to same unit.

1. Introduction

Dynamic load-balancing is a method for distributing the load (data, calculations, etc.) between available resources, like computing nodes or network connections, based on the resources efficiency over the time (described in detail in \cite{1}). Load-balancing algorithms aim to balance the load to prevent coexistence of overloaded and idle resources, as well as to prevent from slowing down the more efficient parts of the system by the slower parts. Such balancing is also important because it helps to increase the fault-tolerance of the whole system, amongst others, by removing single points of failure. In this paper we discuss a load-balancing algorithm for a distributed data acquisition system based on a non-blocking network. We focus on analysing load-metrics and load-communication patterns and their impact on fault tolerance of the whole system.

As a case study we will consider the Data Acquisition (DAQ) System of the Compact Muon Solenoid (CMS) experiment at CERN’s new Large Hadron Collider. CMS is a multi-purpose detector for studying proton-proton and heavy ion collisions at TeV scale \cite{2}. CMS is designed to collect data at the LHC bunch crossing frequency of 40 MHz. The first level trigger pre-selects events with interesting signatures reducing the incoming data rate to a maximum of 100 kHz. The DAQ System
acquires event fragments from about 500 sources called Front-End Readout Links (FRL) and combines them into full events. Each FRL is expected to deliver event fragments of average size of 2 KB at a rate of 100 kHz. As shown on figure 1, the Event Builder is composed of the Front-End- Driver(FED)-builder and the Readout Unit(RU)-builder, which is divided into several autonomous parts called DAQ Slices. Each DAQ Slice has a distributed readout consisting of Readout Units (RU), a set of Builder Units (BU) that are constructing whole events, and an Event Manager (EVM) that supervises the event building process.

![Figure 1. Schematic view of the CMS DAQ System [2].](image)

The first stage of event building is carried out by the FED-Builder: event fragments are transported by the non-blocking network to the surface, distributed amongst DAQ Slices and assembled to super-fragments by RUs. Subsequently, in the RU-Builder, an EVM assigns super-fragments to a BU that builds the complete event.

In case of congestion, back pressure is propagated from the RUs to the FRLs and then to the FEDs. FEDs in turn, in order to avoid buffer overflows, may throttle the trigger rate through the Trigger Throttling System (TTS).

Currently events are distributed statically amongst DAQ Slices using a lookup table that is set at configure-time (usually to implement a round robin algorithm). This method does not take into account variations in DAQ Slice’s processing power over the time which could be caused by faults like failing computing nodes, or broken network connections. As a result, when one DAQ Slice becomes less efficient it slows down other DAQ Slices. Our goal is to propose a load metric and a communication pattern for an algorithm that aims to balance events between DAQ Slices dynamically. The algorithm has to ensure that all fragments of the same event go to the same DAQ Slice.
2. Load-metric

Before the load-balancing can be performed and the load distributed between available resources, first the workload of particular system parts, like computing nodes or network connections, has to be determined. In order to measure the load a unified metric for the whole system has to be established. A load-metric is an important and essential part of every load-balancing algorithm and has to be carefully chosen as it has a significant impact on the algorithm’s overall efficiency.

Over the years many load-metrics were proposed. Theimer et al. [3] suggested a metric based on the fastest response policy. To determine the load of particular load-balancing participant, a load-exchange request is multicast to all potential load receivers. It is assumed that the response time is inversely proportional to the receiver’s workload. Therefore, simply the first receiver who responds is regarded as the least loaded. More common load metrics usually explore the availability of resources more directly. They take into account e.g. CPU queue length, I/O queue length, memory utilization etc. Werstien et al. [4] presented a load-metric reflecting CPU utilization, memory utilization and network traffic. He proposed four-level-hierarchy for evaluating the load of computing nodes: idle, low, normal and high. To assign a load-level to a particular computing node, first the average workload in the cluster has to be estimated. Afterwards, the load of the computing node is compared to the average value and on this basis a load-level is assigned to the node. A different strategy has been introduced by Fonlupt et al. [5], in this approach the load of a processor is measured by the data it owns. The total load of the system is estimated as the sum of load of all processors participating in load-balancing (which in this case are all data owned by participating processors). As previously reported by Branco et al. [6] two groups of load metric can be distinguished: generic and specific ones. Discussed above metrics were examples of generic load indices. Applying such a metric to our system would not result in achieving desired objectives because generic load metrics are designed to rate a single computing node, and not, as in our case, a processing unit, which consists of multiple computing nodes. Although, there are some analogies between above mentioned metric’s requirements and those in our system. Those analogies may be used while creating a specific load-metric dedicated for the discussed system.

FRLs deliver data with 100 kHz frequency, which means that approximately every 10 µs there are about 500 new event fragments that need to be assigned to a DAQ Slice. Sending a single message between computing nodes takes about 10 to 100 µs (depending on the network). As a result, currently it is not possible to calculate the workload and exchange the load-data separately for every event. This operation has to be, rather, made for bigger groups of events in advance. Additionally, the proposed load-metric ideally has to meet following requirements: reflect the efficiency of the processing unit (we propose to estimate the efficiency of a DAQ Slice as the number of events built in a given period of time) and provide information about the occupancy of the readout buffers in the processing units (RU buffers), so the delay due to waiting for free space in those buffers could be avoided.

In a DAQ Slice parallelism is achieved by SPMD (Single Process, Multiple Data) technique, and as a result the workload of a DAQ Slice is directly proportional to the data that it has to process. Therefore the best solution in discussed case is to measure the workload of a DAQ Slice by the data it owns (as proposed by Fonlupt et al. [5]). We propose to initially assign \( n \) events to each DAQ Slice and then, after a given time, measure the load of each DAQ Slice by checking the number of events that still have to be built. A DAQ Slice may accept always, regardless of its efficiency, as many events as can fit into its readout buffers. Therefore, \( n \) has to be equal to the readout buffer size divided by super-fragment size (in the real system the fragment size is variable, and at certain probability may significantly differ from the expected fragment size however at the current stage of our research we do not yet address this problem).

The discussed system is a distributed real-time system and as a result it is difficult to estimate occupancy of all RU’s buffers at once. However, an EVM, as it supervises its DAQ Slice, keeps track of how many events were built, which gives us the information about data still owned by the DAQ Slice as long as we know how many events were assigned to a DAQ Slice originally. The proposed load assignment policy assumes that as many events will be allocated to each DAQ Slice as can fit into
its readout buffers. This way, the most loaded DAQ Slice (the one which owns most data) will be assigned with least load and vice versa. Furthermore, each DAQ Slice will own again the same amount of data, and therefore the whole procedure can be repeated again.

So far, we were able to propose a load metric that meets the requirements of reflecting the efficiency of the DAQ Slices and providing information about readout buffers occupancy. But there still remains the question: when and how frequently to measure the load? Osman and Amar [7] introduced taxonomy for dynamic load-balancing algorithms that distinguish two types of initiation: periodic and event driven. The event driven initiation is usually based on workload observation in a particular computing node and can be sender or receiver initiated.

The measurements in each processing unit have to be done at the same time, so they can be easily compared. On the other hand, the frequency of the load measurements has to reflect the rate of the event flow in DAQ Slices. Therefore, the best solution is to estimate the workload of each DAQ Slice every time one of them becomes under-loaded.

3. Load communication policy
After estimating the workload, the load-data has to be communicated. Again, there are several generic strategies for exchanging load-data. Bubendorfer [8] gave us a compact overview of possible load communication strategies. Polling is the simplest strategy based on asking a chosen computing node directly about its workload. In Broadcasting every computing node broadcasts its load-data to any single node in the system. Group Communication (Multicast) narrows the communication only to a group of subscribers, and therefore reduces the network traffic and addresses only potentially involved computing nodes. There are also few variations of Centralized Collection Agents. In Global Agent policy the load-data are periodically sent to a central agent and then periodically broadcasted to the whole system. In Central Agent policy the load-data updates are not done periodically, but rather event driven. Centex is a combination of the two above mentioned policies. The load-data are collected periodically, but distributed only on event-driven demands. The Worm is an example of totally different strategy. In this policy each computing node communicates its workload only with its neighbours. Moreover, the node also passes the load-data it obtained from his neighbours. These load-updates are done periodically. In this way, the load-data of a particular node is propagated more and more over the whole system with each load-update-cycle.

In the discussed system we have to provide the load communication between each DAQ Slice and the source nodes. We have to keep in mind that also DAQ Slices, themselves, need to communicate in order to start the load-data update at once. For the communication from a DAQ Slice to source nodes, group communication has been adopted. Through a combination of non-blocking network and multicast-facilities we are able to provide an efficient and scalable solution. The idea behind that is to send the load-data in two stages. First, the load-data has to be communicated inside the DAQ Slice. The slice’s network is implemented by Terascale Force10 switch that supports scalable multicasting. Thus, in the first step the load-data is multicast from EVM to RUs. Afterwards, in the second stage, RUs have to pass the load-data further to FRLs. Since the FED-Builder network is a non-blocking network (based on Myrinet technology) RUs may send the load-data fully in parallel without interfering with each other. In this way the sequential communication has been significantly reduced, which is important to achieve good scalability and efficiency.

In terms of DAQ Slice – DAQ Slice communication one-directional ring topology has been studied. For the purpose of this intercommunication a DAQ Slice is represented by its EVM, as it is its central point. In the considered case each processing unit has only one neighbour: the second slice is the neighbour of the first one, the third is the neighbour of the second one, and so on, until the last one, whose neighbour is the first slice. If a DAQ Slice fails for some reason, it is excluded from the circle and its neighbour becomes the neighbour of the previous DAQ Slice. When a DAQ Slice becomes under-loaded it increments its load-update counter and notifies its neighbour, afterwards the newly notified DAQ Slice notifies its neighbour, and so on, until the circle is closed. Each DAQ Slice may inform its neighbour about a given load-data update only once. If a DAQ Slice already triggered load-
update itself, because it became under-loaded, and then was notified by its predecessor it will not pass this notification further. We designed this strategy having in mind that processing units are equally efficient (until some fault occurs). It is very likely that all DAQ Slices reach the under-loaded state and send the notification to their neighbours in the same time, therefore adding an additional DAQ Slice to the system does not introduce additional delay. It can be easily noticed that this strategy is convenient for detecting processing units that for some reason became slower, at a very low network-traffic cost. It is also possible to enhance this method so that DAQ Slices not only notify each other but also exchange load-data. In this way the data redundancy is increased, and therefore greater reliability can be achieved. Naturally, as a result, a DAQ Slice would have to wait for a notification containing load-data from its predecessor before it can communicate the workload to the FRLs.

4. Load assignment method
Now, when the load metric and communication pattern are defined, we can briefly describe the load assignment method. We propose to place the decision procedure in every data source (FRL), and choose the receiver DAQ Slice in each of them asynchronously. The assignment decision has to be based on load-data from all DAQ Slices. A single piece of load-data carries the information about the number of events that should be allocated to a particular DAQ Slice. Each data source has a set of counters corresponding to DAQ Slices. A counter is indicating how many event fragments are allocated to the corresponding DAQ Slice. An event fragment is assigned to the DAQ Slice with the largest counter, then the counter is decremented and the next event fragment is again assigned to the slice with currently largest counter, and so on until all event counters reach 0. At that point, the FRL is starting to use new set of counters populated with new load-data obtained from DAQ Slices.

It can be easily noticed that the proposed method is deterministic and its result depends only on the load-data obtained from DAQ Slices. Moreover, event fragments are produced in the same sequence in all data sources. All of which lead us to the conclusion that all fragments of a single event will be assigned to the same DAQ Slice.

5. Results and Discussion
To prove the correctness of the proposed load metric a prototype was tested in a CMS DAQ test environment. We used a 4 DAQ Slice setup with 1 EVM, 3 RUs and 4 BUs per DAQ Slice. The system was tuned so that the throughput limitation would come from event filter farm and the maximum data taking rate would be 50 kHz (12.5 kHz per DAQ Slice, which is the nominal speed for the production system). The initial trigger rate was set to 50 kHz.

As shown on figure 2 we studied the response of the system to the decrease in processing power in a particular DAQ Slice. In order to simulate this, after a certain period of time Builder Units were killed, one after another. After the 1st BU was killed the standard system lost one quarter of its original capacity. In contrast to that, the system running with the load-balancing algorithm lost only 1/16 of its capacity. It is due to the fact that in the standard system any loss of performance in one DAQ Slice affects other DAQ Slices, and as a result the maximum event building efficiency of each DAQ Slice equals to the maximum efficiency of the slowest one. Subsequently, capacity loss in one DAQ Slice effects in \(m\) times bigger capacity loss in the whole system, where \(m\) is the number of DAQ Slices. On the other hand, while running with the load balancing algorithm the load has been distributed between DAQ Slices accordingly to their performance allowing each DAQ Slice to perform at its maximum capacity. Furthermore, the system running with discussed load-balancing algorithm proved to be more fault-tolerant as a malfunction of one DAQ Slice did not stop further data taking.

As previously mentioned in Section 2, each DAQ Slice is assigned with \(n\) events. As a result in the worst case the system adapts to an imbalance occurrence in the maximum time between two load updates, \(t_{\text{max}} = \frac{m \cdot n}{\text{data taking rate}}\). In the studied system \(m = 1000\) and hence \(t_{\text{max}} = 80\) ms.

The overhead of the discussed workload estimation and communication method has been measured in a CMS DAQ test environment for a system of 8 FRLs (plus 1 FRL for the trigger) and 8 DAQ
Slices, each of them consisting of 1 RU and 2 BUs. The initial trigger rate was set to a value above the capacity of the system. As a result, the system was running at maximum possible data taking rate. The throughput was measured for event sizes in the range from 128 B to 10 kB. Comparison of the results obtained for the standard system and the system running with discussed load balancing algorithm lead as to the conclusion that for event fragment sizes above 2 kB (the operating range for CMS DAQ System) the overhead due to the algorithm was less than 1%.

![Graph showing system response to fault occurrence in one DAQ Slice](image)

**Figure 2.** System response to fault occurrence in one DAQ Slice

### 6. Future work

In the future we intend to enhance the algorithm so it could be also tolerant to faults occurring in RUs. We are also planning to address the problem of variable event size and study the impact of the load assignment policy on the overall efficiency. In the next step we are going to test the algorithm on the full scale CMS DAQ System.

### References

[1] Shirazi B A, Hurson A R and Kavi K M 1995 *Scheduling and Load Balancing in Parallel and Distributed Systems* (Los Alamitos: Wiley-IEEE Computer Society Press)

[2] The CMS colaboration et al 2008 The CMS experiment at the CERN LHC *JINST* 3 261-82

[3] Theimer M M, Lantz K A and Cheriton D R 1985 Preemptable remote execution facilities for the V-system *SIGOPS Oper. Syst. Rev.* 19 2-12

[4] Werstein P, Situ H and Huang Z 2006 7th Int. Conf. on Parallel and Distributed Computing, Applications and Technologies 2006 (Washington: IEEE Computer Society) pp 569-577

[5] Fonlupt C, Marquet P and Dekeyser J 1996 Data-Parallel Load Balancing Strategies *Parallel Computing* 24 1665-1684

[6] Castelo Regina K R L J and Moreno Ordonez E D 2006 *Proc. of the 2006 Int. Conf. on Hybrid Information Technology - Volume 02* vol 2 (Washington: IEEE Computer Society) pp 206-14

[7] Osman A and Ammar H 2002 Dynamic Load Balancing Strategies for Parallel Computers *Sci Ann Comput Sci* 11 110-20

[8] Bubendorfer K P 1996 *Resource Based Policies for Load Distribution* (Virginia University of Wellington) pp 30-3