DISSERTATION

ON A MULTIPROCESSOR COMPUTER FARM FOR ONLINE PHYSICS DATA PROCESSING

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Doktors der technischen Wissenschaften

eingereicht an der Technischen Universität Wien
Technisch-Naturwissenschaftliche Fakultät

von

Nikolaos J. Sinanis

24 Les Erables, F-01710 Thoiry, Frankreich

Matrikelnummer: 9327425

geboren am *27 Mai 1965 in Athen, Griechenland*

Genf, im 23 März 1999

Contents

Contents 1

Kurzfassung 5

Abstract 7

Preface 9

Acknowledgements 13

1 Physics Experiments and Data Acquisition 14

1.1 Elementary Particles and Their Interactions 14
1.2 Particle Physics Experiments 15
1.3 The Large Hadron Collider (LHC) 16
1.4 The CMS Detector 18
1.5 The CMS Trigger and DAQ System 19
  1.5.1 The Level-1 Trigger (LV1) 20
  1.5.2 Data Acquisition Architecture 22
  1.5.3 Event Builder Architecture 22
  1.5.4 High-level Triggers 23
  1.5.5 On-line Computing Services 24
1.6 Summary of CMS DAQ Parameters 24

2 Event Building and Filtering Systems 26

2.1 Event Builder 26
  2.1.1 Read-out Dual Port Memories (RDPM) 27

- 2.1.2 Event Builder Communication Network 29
- 2.1.3 The Switch-to-Farm Interface (SFI) 33
- 2.1.4 Event Manager (EVM) 34

2.2 The Event Filter Farm (EFF) 35
- 2.2.1 Architecture 35
- 2.2.2 Scheduling 36
- 2.2.3 Control and Management 36
- 2.2.4 Alternative Solutions 37

2.3 The Event Filter Unit (EFU) 37
- 2.3.1 Architecture 38
- 2.3.2 I/O Interfaces 38

2.4 Filtering Software. 39

3 System Architectures 41

3.1 Multiprocessor Architectures. 41
- 3.1.1 Symmetric Multiprocessors 43

3.2 Microprocessors 44

3.3 Memories 46
- 3.3.1 Virtual Memory. 46
- 3.3.2 Memory Hierarchies 47
- 3.3.3 Cache Memory Systems 49
- 3.3.4 Memory Models 50

3.4 System Networks 51
- 3.4.1 Bus Interconnections. 52
- 3.4.2 Cross-bar Interconnections 52

4 Performance Aspects and Evaluation Methods 54

4.1 Performance of SMP Systems 54
- 4.1.1 Interconnection Performance 55
- 4.1.2 SMP Operating Systems Aspects 58

4.2 Performance Modeling and Evaluation 61
- 4.2.1 Mathematical Modeling 62
- 4.2.2 Simulations 64
- 4.2.3 Benchmarks and Measurements 66

4.3 Summary 68

5 Farm Simulations 69

5.1 DAQ Simulation Model Description 69

5.1.1 Farm Model. 71

5.2 Input Parameters. 73
5.2.1 LV1-Trigger Characteristics 74
5.2.2 Event Size Distribution. 74
5.2.3 High-Level Triggers Workload. 75
5.2.4 Parameters of the RDPM 76
5.2.5 Event Builder Latency 76
5.2.6 EFU Parameters 78

5.3 Simulation Program 78
5.3.1 CNCL Class Library. 78
5.3.2 Simulator Implementation 79
5.3.3 Output Data Format 79
5.3.4 Model Validation and Verification 80

5.4 Simulation Runs. 80
5.4.1 RDPM Behavior 81
5.4.2 Event Manager 82
5.4.3 SFI and EFU 83

5.5 Summary 89

6 EFU Prototype Environment 90

6.1 SFI Emulation 90
6.1.1 Network layer 91
6.1.2 Fragment Management Layer. 91
6.1.3 Filter API Layer 92

6.2 Setup Description. 92
6.2.1 Event Fragment Data Model. 93
6.2.2 SFI Emulator Implementation 95

6.3 ATM Performance 96
6.3.1 NIC Performance 98
6.3.2 Host Hardware and Software Performance. 101

6.4 SFI Emulator Performance Measurements 102
6.4.1 Contention Analysis 104

6.5 Performance Considerations 107
6.5.1 NIC and Host Related. 107
6.5.2 SFI Emulator. 110

6.6 Summary 112

7 Farm Scaling 113

7.1 Scaling Parameters. 113

7.1.1 Event Builder Size 114
7.1.2 Number of EFU and Processors 115
7.1.3 I/O Bandwidth. 116
7.1.4 Processor Performance 116
7.2 Scaling Scenarios 117
7.2.1 LV1-trigger Maximum Rate 117
7.2.2 Effect of the LV2 Rejection Factor 121
7.3 Conclusion 123

8 Conclusions and Prospects 124

8.1 Impact on the CMS DAQ Architecture 124
8.2 Impact on the EFU 126
8.3 Future Work 127

List of Figures 128

List of Tables 130

List of Acronyms 131

References 134

Curriculum Vitae 140

Kurzfassung

In dieser Arbeit wird eine Untersuchung des Verhaltens von großen Multiprozessor (MP) Computerfarmen für die on-line Datenverarbeitung des Compact Muon Solenoid (CMS) Experiments vorgestellt. CMS ist einer der großen Teichendetektoren am Large Hadron Collider (LHC) am Europäischen Laboratorium für Teilchenphysik (CERN) in Genf, der im Jahre 2005 den Betrieb aufnehmen wird.

Die CMS Computerfarm soll aus 1,000 MP Computersystemen und einem 1,000 × 1,000 Schaltnetzwerk bestehen. Thema der vorliegenden Arbeit ist es mit Hilfe von Simulationsstudien und durch das Erstellen von kleinen Prototypensystemen das Verhalten dieser Computerfarm zu untersuchen.

Für die Simulationsstudien wurde ein einzelereignisgetriebenes Simulationsprogramm entwickelt, welches eine Beschreibung der ‘high-level’ Architektur der Farm, sowie eine Leistungsabschätzung derselben ermöglicht. Die modulare Struktur des Simulationsprogramms erleichtert die Entwicklung verschiedener Module, die das Verhalten der Einzelbausteine in einem gewünschten Detail zu modellieren erlauben. Mit Hilfe dieses Simulationsprogrammes wird eine spezielle Untersuchung über das ‘scheduling’ der Farmknoten durchgeführt, die nachweist, daß ein vorausschauendes ‘scheduling’ die Leistungsfähigkeit der Computerfarm erhöht.

Ein Prototyp eines Farmknotens wurde entwickelt (‘event filter unit’ EFU). Ein System, bestehend aus einem hochleistungs-MP System (dem Farmknoten), verbunden mit einem zweiten Computersystem (die Datenquelle emulierend) über ein ATM Netzwerk wurde erstellt. Das Leistungsverhalten des Anwendungsprogramms im Farmknoten betreffend die Benutzung eines Netzwerk Schnittstellencontrollers (‘network interface controller’ NIC) wird untersucht. Es wird gezeigt, daß ein speziell gebautes Modul zwischen dem Schaltnetzwerk und einem Farmknoten (‘switch-to-farm interface’ SFI) vermieden werden kann durch die Emulation der Funktionen in Software. Darüber hinaus wird ausgeführt, daß eher Aufmerksamkeit erfordert wird für die NIC Hardware, Software und Schnittstelle zum Anwendungsprogramm, als dem Erstellen eines speziell gebauten Moduls für den Aufbau der Ereignisdaten.

Außerdem wird das Erweiterungsverhalten der Farm untersucht. Es wird versucht, bei gegebenen Farm Konfigurationen mögliche Arbeitsbereiche in Abhängigkeit von ver-

schiedenen Netzwerkgeschwindigkeiten zu ermitteln. Analytische Ergebnisse werden vorgestellt, die durch das Simulationsprogramm überprüft wurden. Schlußendlich wird die Abhängigkeit der Anforderungen an die einzelnen Bausteine der Farm von den der Farm eigenen Parametern aufgezeigt.

Abstract

The topic of this thesis is the design-phase performance evaluation of a large multi-processor (MP) computer farm intended for the on-line data processing of the Compact Muon Solenoid (CMS) experiment. CMS is a high energy Physics experiment, planned to operate at CERN (Geneva, Switzerland) during the year 2005.

The CMS computer farm is consisting of 1,000 MP computer systems and a $1{,}000 \times 1{,}000$ communications switch. The followed approach to the farm performance evaluation is through simulation studies and evaluation of small prototype systems—building blocks of the farm.

For the purposes of the simulation studies, we have developed a discrete-event, event-driven simulator that is capable to describe the high-level architecture of the farm and give estimates of the farm's performance. The simulator is designed in a modular way to facilitate the development of various modules that model the behavior of the farm building blocks in the desired level of detail. With the aid of this simulator, we make a particular study on the scheduling of the nodes of the farm, showing that a preemptive scheduling can increase farm's throughput.

We have developed a prototype setup of a farm node —an event filter unit. The setup consists of a high performance MP system (the farm node) connected to a second computer system (used to emulate the data sources) through an ATM network. The performance issues of interfacing a network interface controller (NIC) to the application running in the farm node, are explored. It is shown with the aid of this setup, that the switch-to-farm interface (SFI) —a device used to put together the incoming data fragments into a single entity— can be entirely avoided by emulating its function in software. We show that in order to meet the required event assembly performance in the filter node inputs, the development effort has to concentrate on the NIC hardware, software and its interface to the application, rather than building a custom designed device specialized to perform the task of event assembly.

Finally, the farm scaling issues are investigated. Our aim is to obtain an "operational region" inside the farm configuration space, when the various networking speeds are taken into account. Analytically obtained results that have been confirmed with the above mentioned simulator, are discussed. We present also results showing the influence

of the inherent to the farm parameters (like the algorithm rejection factor) on the requirements for the farm building blocks (sustained I/O bandwidth).

Preface

The quest of the elementary building blocks of matter and their interactions is the research topic of high energy physics (HEP). To find new particles and prove existing theories, one has to collide particles at higher and higher energies. To increase the probability of observing new phenomena, high interaction rates of particles are required. Any evidence of new particles has to be picked out of an extremely large amount of data, recorded during the operation of the physics experiments.

A new generation of HEP experiments at the future Large Hadron Collider (LHC) at the European Laboratory for Particle Physics (CERN), is currently designed and expected to start operating in the year 2005. The LHC is designed to collide protons with interaction energy of 14 TeV, every 25 ns. The LHC experiments have set the objective of investigating the Higgs sector of the Standard Model —the today's theory describing best the microcosm's particles behavior. The Higgs particle according to the Standard Model is the missing piece of the puzzle that explain how particles acquire their mass.

The Compact Muon Solenoid (CMS) experiment at the LHC, besides its many physics and construction challenges, has also an immense on-line data processing challenge. The data acquisition system (DAQ) of CMS is designed to be able to process quickly the detector data produced by the approximately 100 million detector channels after each particle collision, discarding less interesting events and analyzing the most interesting ones. This event selection is achieved by different selection mechanisms, called triggers. The first level trigger selection is expected to reduce the data rate, down to 100 thousand events per second, with an event size of 1 MB. After that first selection, the data of an event must be collected together from the 1,000 read-out units to a single location where further selection will take place. At that high event data rate, many locations where further processing can take place simultaneously, are required. A 1000×1000 switching communication network with an aggregate bandwidth of 500 Gb/s is foreseen for that purpose. It will interconnect the 1,000 read-out units with a computer farm consisting of 1,000 multi-processor (MP) computer systems.

In the event filter farm (EFF) of the CMS DAQ system, the final steps of the event selection will take place before the data can be recorded to permanent storage. For that purpose, the EFF has to provide an estimated total processing capacity of approximately 5 TIPS (5 million MIPS), so that sufficient processing can be done before an event is accepted or rejected. Each of the farm nodes, the event filter unit (EFU), is designed to process an independent stream of events with an average arrival rate of 100 events per second. After a two level processing, the resulting average output rate will be approximately 0,1 events per second. To achieve this high reduction of the event rate, a novel idea of a two-steps data selection has been adopted. At the first step of the selection, called Level-2 (LV2) trigger, only a small part of the processed event (~ 25% of the full event) is forwarded through the switch to an EFU. If the selection algorithm decides that it is an interesting event, the rest of the event will be forwarded to the same EFU for additional processing, the Level-3 (LV3) trigger. This two-steps event selection results in decreased bandwidth requirements for the switch and makes more economic use of both the switch and EFF resources.

Today’s computer performance is not sufficient to build such a farm in an economic way. It is expected however, that prior to the start of operation of the experiment in the year 2005, it will become affordable to build a farm whose performance is very close to the requirements. In that respect a farm approach for the on-line data processing of the CMS experiment, has the advantage of easily adjusting the farm performance, either by selecting more powerful nodes, or by adding more nodes to it. This flexibility offered by a farm design, will ease the building, tuning and upgrade of the CMS on-line event filter farm.

Motivation and Problem Statement

The advent of powerful, yet affordable computer systems, has a tremendous impact also in the data processing required by modern HEP experiments. Building a farm of MP systems for the on-line filtering purposes of CMS is a very challenging task. Design issues, as the number of the event filter units (EFU), the number of processors in each EFU, must be well understood for the various scenarios of the farm operation. These issues are a function of many parameters of the data acquisition system, *e.g.*, the arrival rate of data from the detector, the processing time needed for deciding whether an event will be recorded or not, etc. All that suggests, that a thorough understanding of the farm behavior under many different assumptions is required, before important design decisions are taken.

Specific characteristics of the EFU like their I/O bandwidth and throughput, need also special attention. They are important in order to ensure that the communication requirements with the event builder switch through the switch–to–farm interface (SFI) are satisfied for the given event rates. Given an MP system for the EFU, the available processing capacity must be accommodated with sufficient memory and I/O bandwidth in order to sustain an incoming events stream up to 100 MB/s. More powerful MP systems will require even higher I/O bandwidth, hence a balance of the computing with the I/O capacity is required. Therefore, the impact of MP systems on the total farm size (i.e. the number of EFU) and the ability of MP systems to deliver sufficient I/O bandwidth to the event filtering applications are very important design issues.

Therefore, it is crucial to study in the above context, the functional behavior of the event filter farm foreseen by the CMS DAQ system. In particular, the relations of the farm size with the required processing time relatively to the computer performance, the event filter unit scheduling for the two-steps event selection and the I/O requirements relatively to the farm size.

Thesis approach

In this thesis the above problems are studied in a twofold way. Firstly, for the investigation of the farm behavior under the different assumptions, simulations are used. A simulation tool is developed for that purpose that is able to describe the behavior of the major parts of the DAQ system related to the EFF. With the aid of this tool, a wide variety of scaling scenarios of the farm can be investigated. A particular study of the scheduling behavior of the EFU is done, in order to improve system performance.

Secondly, a study of the performance of an MP system used for an EFU, is done using a prototype setup. A small EFU prototype environment is built using a modern symmetric multiprocessor system and an asynchronous transfer mode (ATM) communication network. An emulation environment of an EFU is developed. With the aid of the setup, the aspects of MP-based EFU can be analyzed.

Thesis layout

In Chapter 1 the concepts of the high energy physics and the modern experimental approach, are introduced. Chapter 2 introduces the domain of this work, describing the various parts of the currently designed data acquisition system of the CMS experiment.

In Chapter 3 we describe the principles of modern MP computer system architectures. In Chapter 4 we analyze the MP performance issues related to the MP architecture, processor-memory interconnection, I/O and the operating system, in order to identify corollaries to the farm performance. A description of the methods used for the MP systems performance analysis and evaluation, is following. We focus on the method of event-driven simulations as the tool for the performance evaluation of the event filter farm.

In Chapter 5 the development of a simulation tool used for the design and evaluation of the CMS event filter farm is described. Based on a simple behavioral simulation model and a set of identified parameters describing the farm, it is able to give the performance parameters of the various components of the farm and of the data acquisition system. The results of a preemptive scheduling are discussed.

Chapter 6 contains a case study of using a modern high-end commercial computer system as the event filter farm node. The interaction between the computer hardware, the operating system and the communications network, is studied in the framework of an SFI emulator running into the system.

In Chapter 7, we investigate various scaling scenarios of the farm. An assessment of the farm performance scaling is done, for a farm with many but less powerful processing nodes and a farm with more powerful, but fewer processing nodes. The merits of these two possible approaches are studied and compared.

Finally, in Chapter 8 the conclusions of this work are presented. Prospects and future work are discussed at the end.

Achievements of the Presented Work

With the aid of the event filter farm simulator, it is shown that when scheduling of the event filtering jobs is done, the throughput of the farm can be improved. In particular, if preemption of LV3 jobs is adopted when LV2 jobs arrive and the EFU processors are busy processing LV3 jobs, the farm throughput is increased.

The developed simulator also sets the framework for further, more detailed, investigations on the farm architecture. In particular, it can be very useful to extrapolate the performance of prototype systems built during the design phase of the farm, up to the expected performance of a full sized farm.

The EFU prototype setup has given valuable insight on the usability of MP systems for the EFU. In particular, it is shown that an MP system can also be used to perform the functions of the last stage of the event assembly, prior to the event processing. This task, done by the SFI in the original DAQ design, can be effectively emulated in an MP-based EFU. It is shown that the potential performance bottlenecks might appear in the I/O subsystem of an MP system and not related to any limitations of the SFI-emulator’s event assembly throughput. While a custom designed SFI may guarantee the necessary event assembly throughput, the possible I/O performance bottlenecks are still not addressed. The experience gained with this EFU prototype, suggests that in both cases of a custom designed SFI system, or an emulated one, the available application interfacing to the I/O through the operating system, requires a significant improvement. Possible ways of how this could be done are proposed.

Acknowledgements

This thesis would never have been finished without the support of many friends and colleagues. Foremost, I wish to thank my Supervisor, Univ. Prof.-Dr. Günter Haring, for his tolerance and comprehension. He has been extremely supportive throughout this work and through the difficulties I have encountered.

I am grateful to Werner Jank, Sergio Cittolin and Martti Pimiä of CERN, for their continuous encouragement and giving me the opportunity to start the work on this dissertation.

I am obliged to the Institut für Hochenergiephysik der Österreichische Akademie der Wissenshaften in Vienna, for offering me the first opportunity to work at CERN on the CMS project.

I am indebted also to Jean-Pierre Porte for his continuous support and the numerous helpful discussions, to Norbert Neumeister for his help with German and for his friendship.

Special thanks go to Dirk Samyn, who kindly provided me the original version of the event filter farm simulator, to Gerhard Walzel for the many fruitful discussions, Paul Porth for his hospitality, Joop Joosten for the enlightening discussions on ATM and George Smyris for his help on proofreading.

This dissertation has been made possible by the generous Doctoral Students program of CERN, from 1994 to 1997. My work and stay at CERN has been also supported by the management of the CMS experiment, to whom I am very thankful.

The Interphase Corporation has kindly provided the sources of their ATM driver, giving invaluable insight for the ATM implementation of the SFI emulator.

My fiancée Tina, was a source of continuous encouragment and silently provided the necessary understanding for the endless hours in front of the computer.

To all those that helped me, a big thank you!

1 Physics Experiments and Data Acquisition

The mankind endeavor to understand the fundamental laws and principles of Nature dates back to the days of Democritus, Aristotle and Pythagoras. The dramatic evolution of technology and the remarkable intellectual progress of Science, especially in the last century, still suggest that Nature is governed by simple laws, even though the complexity of the observed phenomena might imply the opposite.

The modern science of elementary particles is concerned with the identification of the fundamental building blocks of matter and their interactions. In numerous achievements it has revealed a fascinating harmony of Nature, is it at the macrocosm or at the microcosm, which today is very well described by the *Standard Model* (SM) of elementary particles and their interactions.

1.1 Elementary Particles and Their Interactions

The different forces of Nature as described by the field theory, are four: the *electromagnetic,* the *weak*, the *strong* and the *gravitational* force. Each fundamental interaction has its own field and one or more mediator particles, responsible for the exchange of the interaction. These fundamental forces and their mediator particles are summarized in Table 1.1, together with their relative strength and effective range.

Force	**Mediator**	**Relative Strength**	**Range**
Electromagnetic	γ	*1*	∞
Strong	g	~ 20	~ 10^{-15} m
Weak	$W^{\pm}, Z^0$	10^{-7}	~ 10^{-18} m
Gravity	*graviton?*	10^{-36}	∞

Table 1.1: Fundamental forces and their characteristics

The remarkable success of the SM is that it can describe the interactions of the various particles discovered in the numerous physics experiments during the last century, making use of only three families of pairs of elementary particles (leptons and quarks) and their anti-particles, by unifying the electromagnetic and weak forces into the *electroweak* force. Those two groups of three families of particles are summarized in Table 1.2.

	Family		
Leptons	$\begin{pmatrix} e \\ \nu_e \end{pmatrix}$	$\begin{pmatrix} \mu \\ \nu_\mu \end{pmatrix}$	$\begin{pmatrix} \tau \\ \nu_\tau \end{pmatrix}$
Quarks	$\begin{pmatrix} u \\ d \end{pmatrix}$	$\begin{pmatrix} c \\ s \end{pmatrix}$	$\begin{pmatrix} t \\ b \end{pmatrix}$

Table 1.2: Fundamental elementary particles

In order to explain the spontaneous symmetry breaking in the electroweak force, into what is observed today as electromagnetic and weak forces, the SM introduces a mechanism which also gave masses to the light and heavy particles and to the vector bosons $W^{\pm}$ and Z^0. This mechanism requires the introduction of another particle; the *Higgs boson*. In further extensions of the SM as in the *Minimal Supersymmetric SM*, the Higgs boson is replaced by a set of five bosons $H^{\pm}$, h, H^0 and A.

1.2 Particle Physics Experiments

The SM has successfully predicted the existence of the $W^{\pm}$ and Z^0 intermediate bosons, with a remarkable precision of their masses. For their discovery, the collision products of protons p and anti-protons $\bar{p}$ had to be registered, identified and analyzed. In 1981 at the European Center for Particle Physics (CERN) in Geneva, Switzerland, the Super proton anti-proton collider ($\mathrm{Sp\bar{p}S}$) was constructed, and the detectors UA1 and UA2 built around it confirmed the SM predictions.

To study further the properties of the intermediate bosons $W^{\pm}$ and Z^0, another generation of physics experiments were constructed at CERN at the beginning of 90's. Four world-wide collaborations were formed and a new collider of electrons and positrons (LEP) was built, together with four experiments (L3, ALEPH, OPAL and DELPHI) around it. Other experiments (CDF, D0) have been setup at the Tevatron collider at Fermilab, in the United States.

It was the LEP experiments that confirmed the assumption of the SM that the leptonic families are exactly three. Also, the CDF experiment [CDF96] at Fermilab, has recently provided evidence of the existence of the last quark remained not detected; the *t*-quark.

All these searches of rare interactions of particles, able to prove the existence of one or another particle, are not only major scientific and engineering challenges, but also enormous data processing and analysis projects. High-speed electronics hardware and sophisticated physics software, are all put together to select only the interesting candidates of the registered interactions (from now on referred to as events), out of a huge number of background and uninteresting events. There is an immense need in High Energy Physics experiments, of high performance computing resources and communication systems.

1.3 The Large Hadron Collider (LHC)

The existence of the Higgs boson still remains unconfirmed in the theory of the SM. For that purpose, at CERN, the physics community has proposed to build the Large Hadron Collider (LHC) [LHC95].

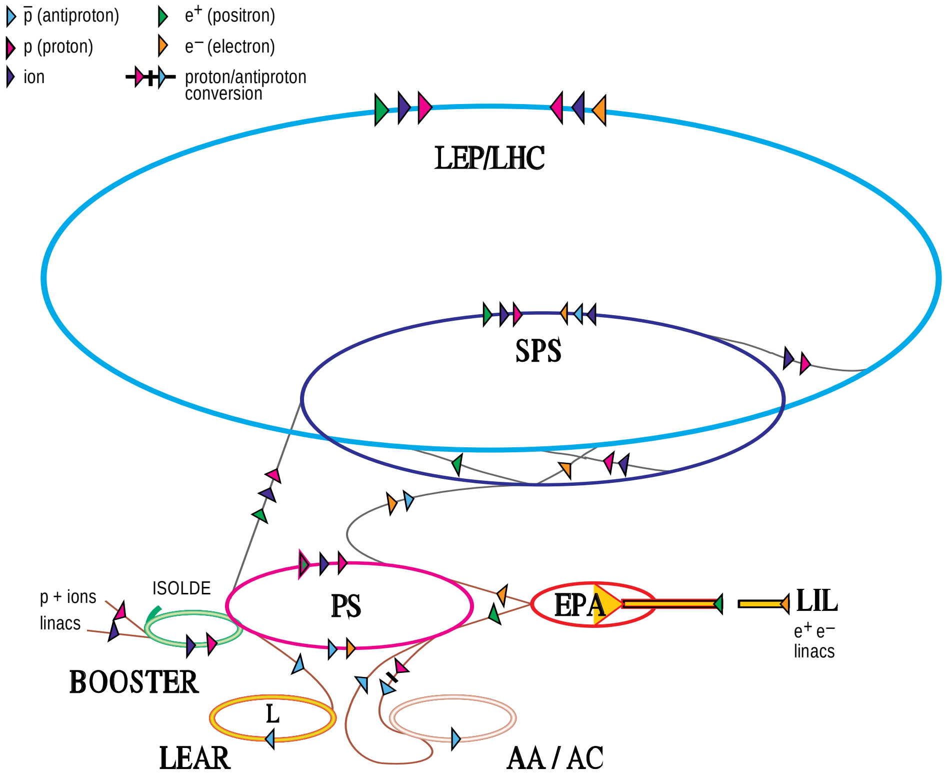

Figure 1.1: CERN accelerator complex

The LHC will accelerate two beams of protons in opposite directions and collide them at a center of mass energy of 14 TeV/c^2. The construction of LHC was approved in 1994 and it will be operational in 2005.

Two general purpose experiments, CMS [CMS94] and ATLAS [ATLA94], have been proposed to be built at the LHC, to search for the Higgs boson and any other interesting physics phenomena that might emerge at the unprecedented energy levels of LHC. Two other more specialized experiments, LHC-B and ALICE are also proposed.

The LHC will be built in the same tunnel where the LEP collider is located now. It will have a circumference of 27.5 km. In order to achieve the required collision energy at LHC, the existing accelerator complex of CERN consisting of the LINAC, the Booster, the Proton Synchrotron (PS) and the Super Proton Synchrotron (SPS), will be utilized to gradually accelerate the collision particles and finally inject them into LHC (Figure 1.1).

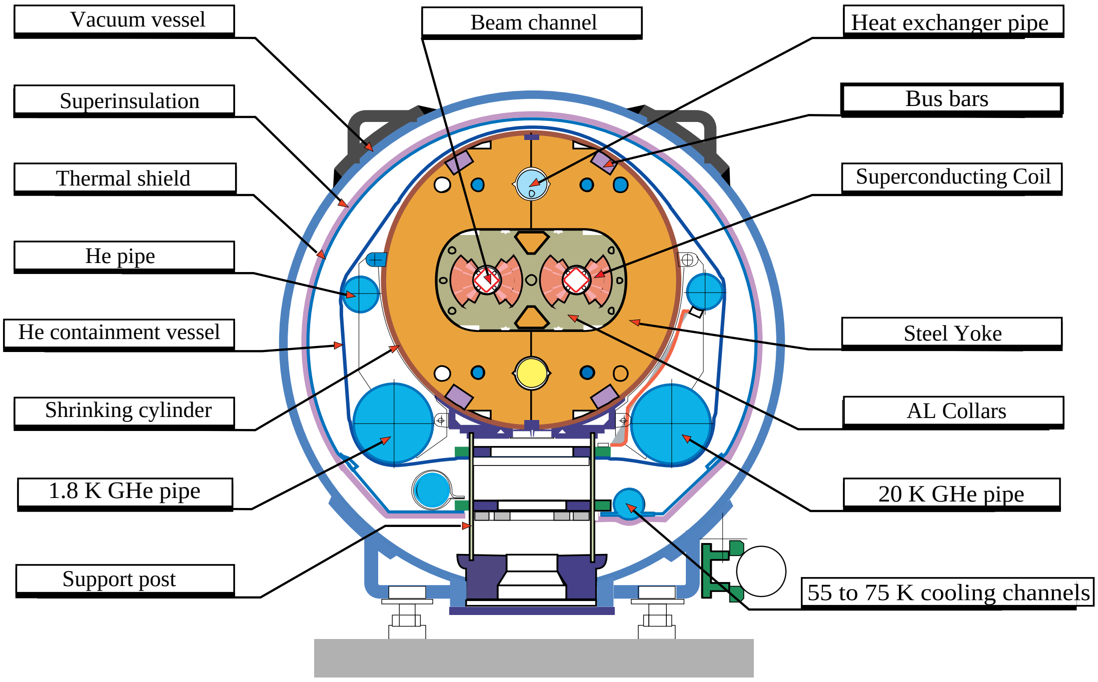

Figure 1.2: Cross-section of the LHC dipole

The maximum energy that LHC can achieve, depends on how strong the magnetic field that holds on trajectory the collision particles can be made. In order to achieve the required energy of 7 TeV for each beam, some 1300 dipoles (Figure 1.2) with superconductive magnets generating a magnetic field of 8.4 T strength, will be utilized.

At normal operation, LHC will collide bunches of protons every 25 ns, each bunch crossing producing approximately 20 proton interactions. Two periods of operation are foreseen for LHC. LHC will provide also the capability of colliding heavy-ions at lower energies, and the ALICE experiment is specially designed to observe these collisions.

1.4 The CMS Detector

The Compact Muon Solenoid (CMS) detector, depicted in Figure 1.3, was first proposed in 1990. Its basic concept for detecting the Higgs boson, is a compact detector with optimized muon identification and the best possible electromagnetic calorimeter.

The CMS detector consists of the pixel detector, the inner tracker, the electromagnetic and hadron calorimeters and the muon detectors. A solenoidal magnet generating a field of 4 T intensity, surrounds the central detectors and the calorimeters.

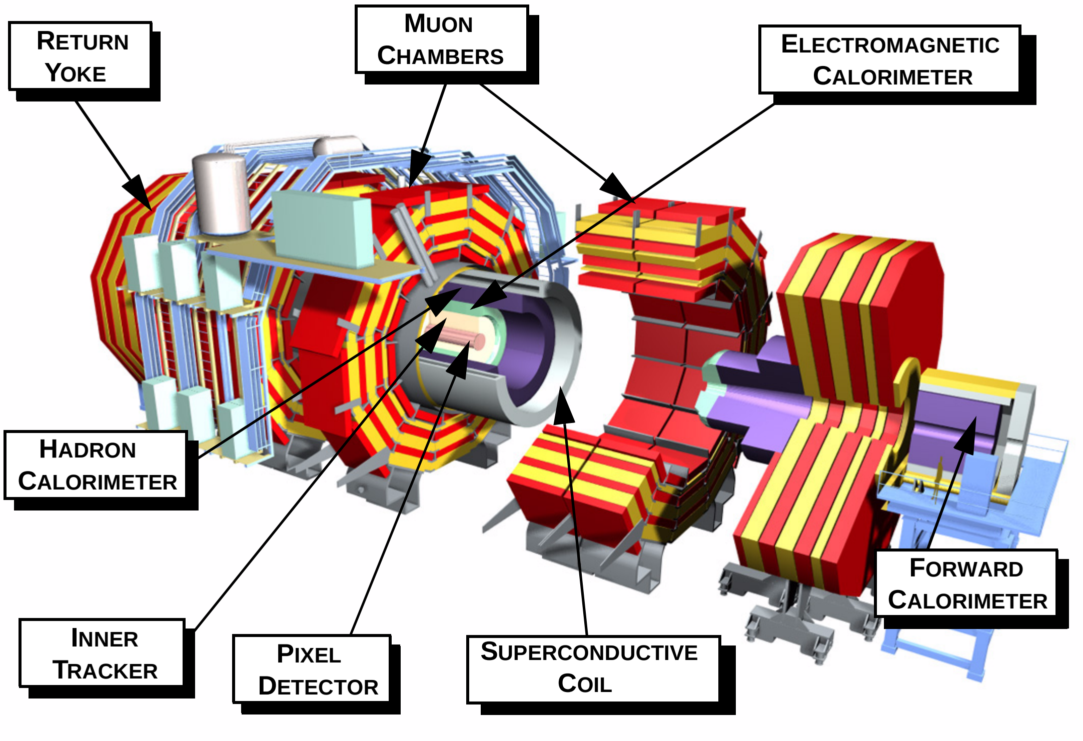

Figure 1.3: Perspective view of the CMS detector

The CMS detector has a total length of 21.60 m, a diameter of 14.60 m and it weighs 14,500 tons.

Until the beginning of construction of CMS, several research and development projects on detector technology are taking place, in order to evaluate the different choices for each sub-detector system. In addition, the requirement for an affordable detector will lead to two designs, a full-fledged and a staged design, with the option of future upgrades. The CMS detector performance and objectives are such, that it can provide a rich physics research program for at least 15 years.

Today the CMS project is supported by a world-wide collaboration, consisting of 1760 scientists in 145 universities and research organizations, from 31 countries. The CMS project was approved in December 1996, with a construction budget not exceeding 475 million swiss francs. Its targeted completion date is in year 2005.

1.5 The CMS Trigger and DAQ System

The LHC experiments' objective of detecting the Higgs boson, raises the requirement of operating the LHC at high luminosity *i.e.*, the LHC provides a high number of interactions per unit of time, so that rare phenomena can be detected.

With bunch collisions occurring every 25 ns, the CMS detector will record the traces of approximately 20 proton–proton interactions at a frequency of 40.8 MHz. A large number of sub-detector channels are therefore required, to keep the number of “fired” detector channels (referred here as occupancy) at a reasonable level. For every occurring proton collision, the particles produced will be detected by the individual sub-detector systems. The output data of each sub-detector, after a collision has taken place, will add up to the full event describing the results of the interaction. In Table 1.3 are summarized the contributions of each sub-detector system to the full event, together with their estimated occupancy. The total size of one event is expected to be around 1 MB of compressed (zero suppressed) data.

Detector	**Channels**	**Occupancy**	**Event Size [KB]**
Pixel	*80,000,000*	*0.01*	*100*
Inner tracker	*16,000,000*	*3.0*	*700*
Preshower	*512,000*	*10.0*	*50*
Calorimeters	*125,000*	*5.0*	*50*
Muons	*1,000,000*	*0.1*	*10*
Trigger data			*10*

Table 1.3: CMS detector channels and event sizes

If we assume that each one of the detector channels carries binary information and multiply the interaction rate by the resulting (from the detector channels) number of bytes, we reach to an estimate value of 400 TB/s for the required data throughput. The effect of using an event size of 1 MB reduces the above figure by an order of magnitude only (40 TB/s). Therefore, before the event rate can be handled for storage, a multi-stage event selection and rate reduction mechanism is required to reduce the data rate to reasonably lower figures. This selection however, must not compromise the interesting physics events that can occur. These event selection mechanisms are commonly referred to in the physics experiments, as *triggers*. Their implementation is usually based on specialized, custom-made hardware.

The CMS DAQ system implements three logical levels of triggering. The first data selection is carried out from the LV1 trigger devices. Two additional stages of event selection will be implemented by the LV2 and LV3 triggers respectively, before the resulting event rate becomes low enough for data storage.

1.5.1 The Level-1 Trigger (LV1)

The LV1 trigger [Lack95] consists of specialized hardware, able to process each subdetector's data in a very limited time interval (few bunch-crossing periods each of 25 ns), and provide an indication if the occurred event is interesting or not. Every sub-detector participating in the LV1-trigger has its own trigger devices because of the peculiarity of their signals and the different logic needed for a trigger decision. Because the varying response time of the different sub-detectors, not all of them are suitable for LV1 triggering. There are also specialized sub-detectors that have a fast response time and are used only for triggering.

When the various LV1 trigger devices have finished processing, a global decision must be taken to forward the event for further processing or not, according to the result of the individual LV1 triggers. This decision is taken by the Global Trigger System (GTS) [Neum97]. Among its tasks of combining the triggers, the GTS must be able to pace the resulting event rate. The average rate of events accepted from LV1 is fixed today to a maximum value of 100 kHz. The exact characteristics of this event rate are not currently well known. The regional LV1 triggers may result to accepted events in subsequent bunch-crossings *i.e.,* 25 ns. The GTS is expected to minimize such cases to not more than three triggers in consecutive bunch-crossings.

The GTS is designed to be programmable in order to accommodate unforeseen physics rates and to operate in a pipeline mode due to the high speed and data-flow requirements. The acceptance of an event by the GTS, will be communicated by the timing, trigger and control (TTC) system [Tayl95] to the front-end detector electronics, to initiate the detector read-out.

The identification of the event that a Higgs boson might have occurred, is done through the various signatures of the typical Higgs production decays. Each decay, leaves to the sub-detectors distinct signatures that can be identified. These signatures are summarized in Table 1.4 together with their contribution to the total LV1-trigger event rate.

Each trigger condition might occur not only due to a Higgs boson decay, but also from other uninteresting interactions. In a 25 ns time interval, not enough processing can be done to decide whether a Higgs boson was present or not. Therefore, the LV1 trigger will fire only when a particular trigger condition has been detected, while detailed processing that might reveal interesting physics events will be done at the later LV2 and LV3 trigger stages. This way, fast selection of potentially interesting events can be done, resulting in highly reduced selected event rates.

Trigger Condition	**E_t cut-off [GeV]**	**Rate [kHz]**
E_t sum	*400*	*6*
E_t miss	*80*	*4*
e single	*25*	*6.84*
e double	*12*	*1.45*
1 jet	*100*	*2.06*
2 jet	*60*	*2.17*
3 jet	*30*	*3.16*
4 jet	*20*	*2.96*
jet + e	*50 + 12*	*1.35*
Single μ	*20*	*7.8*
Double μ	*4*	*1.6*
μ *+ e*	*4 + 8*	*5.5*
μ *+ jet*	*4 + 40*	*0.3*
μ + *E_t miss*	*4 + 60*	*1.0*
μ + *E_t sum*	*4 + 250*	*0.2*
SUM		*20*

Table 1.4: LV1 trigger rate break-down

The detector front-end electronics, the LV1 trigger and the event readout units, are designed for parallel and pipeline operation. The TTC system is used for their synchronization. It provides a reliable means of signal and short message distribution, between the various stages of the front-end electronics and the LV1-trigger devices.

The TTC system employs a time division multiplexing of two data channels at 160.32 MBaud. One channel is used to communicate the LV1 trigger acceptance and the other for broadcasts and individually addressed commands. TTC offers a high precision clock of 160.32 MHz (four times the LHC bunch-crossing rate), distributed to the various detector destinations with an output jitter of less than 10 ps RMS.

1.5.2 Data Acquisition Architecture

The GTS acceptance of an event starts the read-out of the detectors into some 1,000 front-end buffers. The task of the data acquisition system (DAQ) is to move the data fragments from these front-end buffers, to a single location in the event filter farm (EFF), where the LV2 and LV3 triggers will be executed.

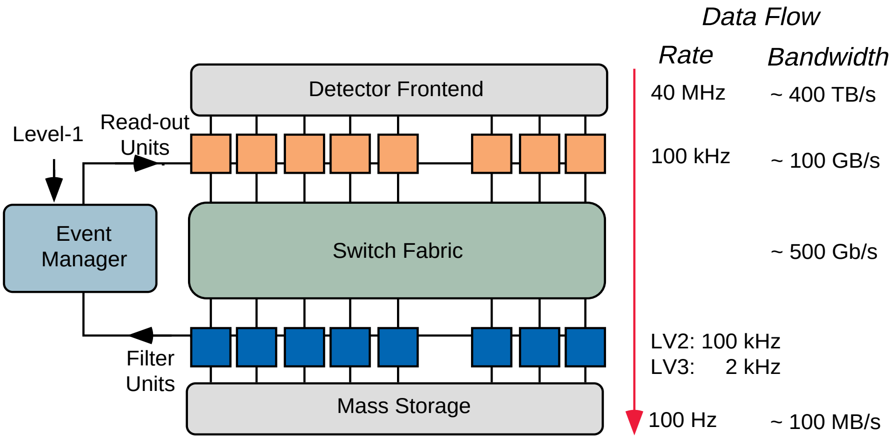

Figure 1.4: Architecture of the CMS DAQ system

The management of the readout buffers is done by the read-out units (RU) which comprise a set of front-end drivers (FED) connected to a read-out dual port memory (RDPM) [Citt95]. The RU will be coordinated by the event manager (EVM) in order to send the data fragments of an event to a single event filter unit (EFU). The task of putting together event data fragments into full events is called *event building*.

This architecture of the CMS DAQ system is shown in Figure 1.4. It is estimated that some 1,000 EFU will be required to handle the LV1 event rate, providing sufficient computing resources to execute the physics algorithms of the LV2 and LV3 triggers.

1.5.3 Event Builder Architecture

The crucial part of the CMS DAQ system is its event building architecture. It is foreseen to incorporate a high performance communication network of 1,000 inputs and 1,000 outputs. However, the bandwidth requirements for such a communications network are very high due to the LV1-trigger data rate and the average event size. The product of the event size times the LV1-trigger rate, gives the bandwidth that the event builder will have to sustain. With a LV1 trigger rate of 100 kHz and an event size of 1 MB (as shown in Figure 1.4), one can see that an aggregate bandwidth around 800 Gb/s will have to be sustained by the event builder.

To require the existence of such a high performance for the event builder, well in advance of the start-up of the CMS experiment, is a very risky assumption. Therefore, in order to reduce the required bandwidth, the event builder architecture is based on a two steps event building, otherwise called *virtual-LV2* trigger strategy. For the LV2 trigger, the physics algorithms will be given to process only a fraction (~ 25%) of the full event. Only in the case of an event accepted by the LV2 trigger the rest of the event (~ 75%) will be forwarded to the same destination, so that the LV3 trigger processing can start. The LV3-trigger will have available the full event for processing.

Given a relatively high rejection rate of the LV2 trigger, significant bandwidth reduction can be achieved. For instance, given that the rate of events accepted by the LV2-trigger will be on average 2,000 events per second, for the above mentioned event fractions the resulting bandwidth that the event builder will have to sustain is around 250 Gb/s. With a safety factor of two the required bandwidth is set to be around 500 Gb/s (Table 1.5). The virtual-LV2 strategy assumes that with the data of the muon detectors and the calorimeters, an efficient decision (without missing interesting events) can be taken by the LV2 trigger.

Constructing a switched, non-blocking and a packet-loss free communication network of the size required by the CMS DAQ architecture is a difficult task. The required performance and the cost considerations suggest that widely-available, highly-scalable commercial solutions are the best candidates for the event builder. Standardized communication technologies as Asynchronous Transfer Mode (ATM), Fibre Channel (FC), Scalable Coherent Interface (SCI) are among the candidates considered today. It remains to be seen which technology will be the market’s choice, eventually offering the best price performance, before a decision is taken for the implementation of the event builder in CMS.

1.5.4 High-level Triggers

In addition to the LV1 trigger, CMS foresees two more levels of event processing (the LV2 and LV3 triggers) in order to select only the interesting physics candidates. The high-level triggers will not be executed in specialized hardware devices as in the LV1-trigger case, but in one of the EFU of the event filter farm which consists of computer systems made of general purpose processors.

The LV2 trigger, in more time-relaxed conditions, will be able to refine the LV1 decision identifying the interesting physics events and discarding the less interesting. The LV2 trigger must have bounded execution time, because a timely decision for each event will be required to release the pending event part in the RDPM.

The LV3 trigger, will make an even narrower event selection based on data from all the sub-detectors. The origins of the tracks left by particles that so far have triggered an interesting event, will be identified among the tracks of many thousand other particles. If there is still an indication that an interesting event has happened, the full event together with the individual trigger data will be stored into permanent storage for later detailed analysis.

The flexibility of general purpose computer systems in the EFF, will help for the tuning of the high-level trigger algorithms. The algorithms' performance and efficiency will be crucial for the operation of the detector. It is not desired that interesting events will be rejected, due to any algorithm inefficiencies. On the other hand, the limitations of the storage devices will set an upper limit to the amount of data that will be possible to be stored.

Today's estimates of the combined processor performance that will be available when CMS will start operating and the amount of processor time needed for the high level triggers, suggest that around 10ms for LV2 and around 1s for LV3 will be sufficient.

1.5.5 On-line Computing Services

At the endpoint of the DAQ system, the events that passed all the triggering stages will be recorded to permanent storage, so that later can be carefully analyzed and the underlying physics interaction studied. The output event streams from all the EFU will terminate at a buffering device. Estimates show that with an output event rate around 100 Hz, a minimum of 10 TB buffer storage capacity will be required. A short *quasi on-line processing* will follow immediately after and prior to the recording [CMS96]. The task of the quasi on-line processing will be to tag the events in the event database used for the off-line analysis.

Other tasks performed by the on-line computing services are the DAQ control and monitoring and detector calibration. The control system supervises the operation of the entire trigger hierarchy and the DAQ system. It will monitor and record the configuration and operation parameters, including initialization and dynamic reconfiguration, and accordingly launch and terminate the DAQ tasks. Failure detection is a crucial task of the control system. Some additional monitoring will be also provided sampling the detector physics performance and detecting error conditions.

1.6 Summary of CMS DAQ Parameters

In Table 1.5 we summarize the main parameters of the CMS DAQ system, as they are assumed today. The value for the bandwidth of the event builder switch is calculated by adding the needed bandwidths for LV2 and LV3, in order to sustain the transfers of a LV2 (LV3) event at a LV2 (LV3) rate and taking into account a 50% switch efficiency. The assumed value for the LV2 rejection rate is set to be 50. One could argue that this is a rather optimistic value. For the performance evaluation studies of Chapter 6 and Chapter 7 we consider worst cases of the LV2 rejection factor value.

Number of RDPM	*1,000*
Number of EFU	*1,000*
Maximum LV1 trigger rate	*100 kHz*
Average total event size	*1 MB*
Size of data used in LV2	*250 KB*
Size of data used in LV3	*750 KB*
LV2 trigger rejection rate	*20 to 100*
LV3 trigger rate	*2 kHz*
Aggregate switch bandwidth	*250 Gb/s*
Accepted events output rate	*100 Hz*
Typical LV2 processing time	*1 ms*
Typical LV3 processing time	*1 s*

Table 1.5: Summary of the CMS DAQ parameters

2 Event Building and Filtering Systems

The event filtering and the event building are the main tasks of the CMS DAQ system in order to accomplish the acquisition of the detector data at a reasonably low rate. Both together, set the domain of this work.

In this chapter, the functionality of the various units comprising the event building and filtering system is described. The basic concepts guiding the design of the CMS event builder and event filter are analyzed. Similarities of the DAQ sub-systems with those used in computer and communication industry, are identified.

2.1 Event Builder

The event builder (EVB) has to assemble the data fragments constituting an event, as recorded by the different sub-detectors, into a single destination—location— in the event filter farm (EFF). This functionality is depicted in Figure 2.1.

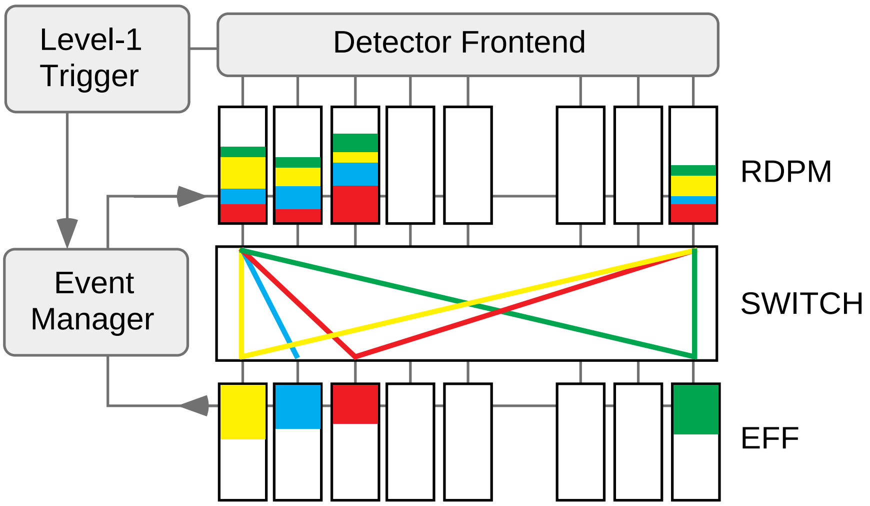

Figure 2.1: Event building using a switch network

The pieces with different shades at the switch inputs (the top side), are representing fragments belonging to different events. The fragments of each event (those with the same shade and spread over the switch sources), will have to be forwarded to a previously defined destination in the EFF.

The event builder has always been the principle component in most DAQ systems of HEP experiments, due to the nature of the data produced by the particle detectors. The generation of the LHC experiments however, due to the high interaction rates and massive data flows, has requirements on the EVB that have not been previously met. A large number of EVB input sources (~1,000) are required for reading-out the detector data. A similar number of destinations or event filter units are needed to handle the resulting event rate.

2.1.1 Read-out Dual Port Memories (RDPM)

The RDPM [Citt95] are hardware devices that interface the EVB with the detector read-out electronics and their front-end drivers (FED). They have as their task the read-out of FED connected to each of them and on request inject the fragments of an event into the EVB switch.

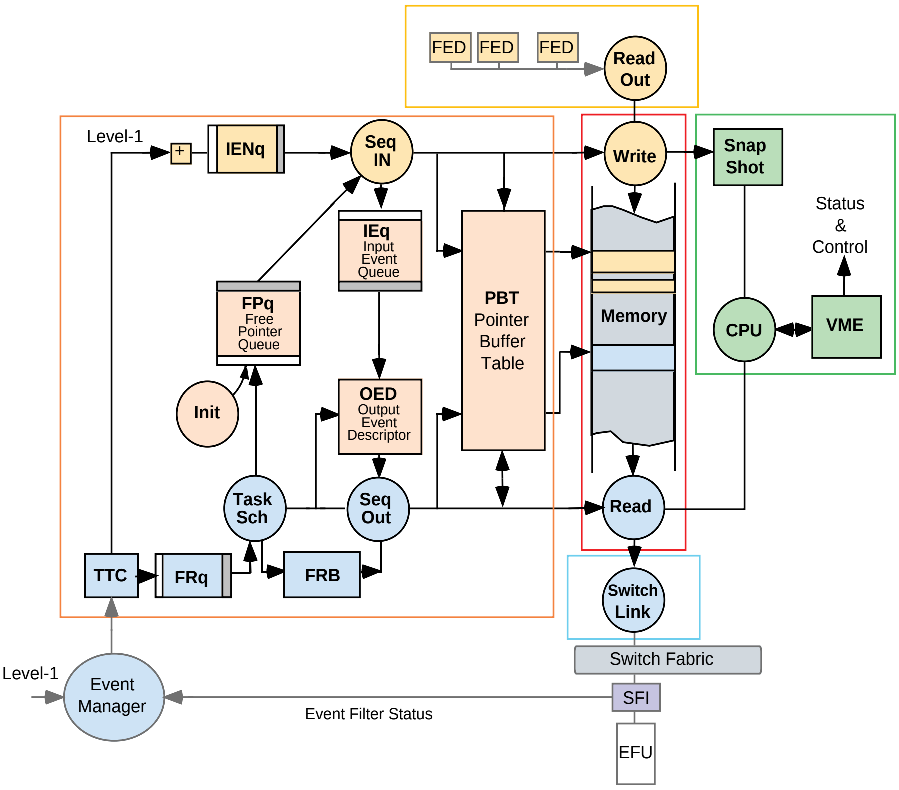

Figure 2.2: RDPM functional diagram

The RDPM essentially represents a large storage device with two simultaneously accessed ports, one used for input and the other for output. The RDPM it is connected also to the LV1-trigger signal distribution network (TTC) and the event manager (EVM). Figure 2.2 shows a block diagram of an RDPM based on FPGA logic [Fucc95].

The RDPM data memory is partitioned into fixed-size pages. A list of pointers to the free memory pages is maintained in the free pointer queue (FPq). The input part of the data stream is handled by the input sequencer (SeqIN) and the output part, by the output sequencer (SeqOUT).

The LV1-trigger sends to the RDPM the event numbers of the accepted events. The RDPM store the incoming event numbers in the input event number queue (IENq). Once data are available in the FED, the RDPM will be notified and the SeqIN will pick an entry from the IENq. The FPq FIFO-organized queue is read to obtain a pointer to a free memory page. The read-out of the FED is then initiated and the incoming data are stored into one or more memory pages. At the end of the read-out an event descriptor consisting of the pointer to the first memory page, the event number and the total word count, is written to the input event queue (IEq). Finally, the SeqIN writes into the pointer buffer table (PBT) a linked list of pointers to the memory pages occupied by the event fragment.

The RDPM output is driven by the EVM. The EVM sends to each of the RDPM event filter requests (EFR). An EFR consists of a filter resource (FRS), the event number assigned to it and a command specifying the action to be performed for that event. Each RDPM will store incoming EFR in its farm request queue (FRq).

When an EFR becomes available, the task scheduler (TS) copies an available event descriptor from the IEq to a free location in the output event descriptor (OED). The TS will also decode the command contained in the EFR and will initiate an action taken by the SeqOUT, depending on the decoded command.

The commands that are envisaged to be encoded in an EFR are READ-data, CLEAR-data or both READ and CLEAR. The CLEAR-only command will be executed immediately, while the other two commands will be written to the filter request buffer (FRB). The SeqOUT on a reception of a command will read the pointer to the first memory page and the word count from the event descriptor contained in the OED. It will then initiate the transfer of the data referenced by the pointers, to the data-link device.

The presence of the FRB is to allow for a higher level of traffic-shaping of the RDPM output data stream to the EVB switch. We will examine how this might be implemented in the next paragraph.

Total number of RDPM	*up to 1,000*
FED per RDPM	*up to 6*
Minimum I/O bandwidth	*100 MB/s*
Maximum I/O bandwidth	*400 MB/s*
Average event-fragment size	*1024 ÷ 4096 bytes*
Number of buffered event fragments	*≈ 100,000*
Internal memory size	*≈ 100 MB*

Table 2.1: RDPM main characteristics

The characteristics of the RDPM as included in their current specifications, are summarized in Table 2.1 The maximum figures for the I/O bandwidth are set to enable the RDPM to operate in event builder environments with eventually smaller switch configurations *e.g.,* 512 × 512 ports.

Similarly to the FPGA based RDPM, there exist proposals of building it with specialized processor boards built around the TMS 320C80 digital signal processor. Such a system is VORTEX [Bran95]. Prototypes of such systems are already built, which come very close to meeting the RDPM requirements.

For the purposes of the virtual-LV2 trigger operation, the RDPM will be grouped in those holding event fragments to be used for the LV2 trigger and those that hold event fragments that will be requested in the case of LV3 trigger. Both groups will get data from their attached FED simultaneously and at a maximum rate of 100 kHz.

The LV2 group of RDPM will receive EVM requests to send data at an average rate of 100 kHz. The LV3 group will receive the read requests from the EVM at an average rate which will be determined, by the average number of accepted events at the LV2 trigger stage in one unit of time. On the other hand, the RDPM of both types (LV2 and LV3) are designed with a predefined memory size. Hence, the amount of time that an event fragment is kept in the RDPM is very critical for their stable operation. Evidently, this hold time is even more crucial for the LV3 group of the RDPM, as it in addition includes the time necessary for the LV2 trigger processing.

The RDPM need to be monitored to ensure their normal operation. This will be done through a VME interface. The same interface will be used for their control and initialization, as well as for any other signalling, required by the DAQ architecture.

2.1.2 Event Builder Communication Network

The heart of the event building task is the communication network that interconnects the read-out units with the event filter units. The CMS DAQ architecture, foresees a large switching network to accomplish this task. However, the number of needed ports, the

LV1-trigger rate and the average event size, result in non-trivial requirements on the EVB network.

Preliminary estimates indicate that around 1,000 input ports will need to be connected to a 1,000 output ports. The number of the input ports is determined by the number of the read-out units that are necessary to handle the detector data. Consequently, the number of the output ports is determined by the number of the required EFU, in order to provide sufficient event processing resources, to handle the LV1 and LV2 event streams and rates. As was discussed in paragraph 1.5.3, the EVB switch will have to sustain an aggregate bandwidth of 500 Gb/s.

There are many similarities between an EVB switch and those used in telecommunications and high-speed networking [Bars90]. The switch design will benefit a lot if commercially available equipment can be used. This will lead to much more economical solutions than in the case of a custom design. The main differences between a switch suitable for event building and a telecommunication switch, are on the assumed patterns of traffic. In telecommunications, random input–output interconnection patterns are assumed usually. Traffic models for high-speed data networking assume random traffic, or data arriving in packet trains. In event building however, the interconnection patterns are a sequence of input-output combinations that might be known beforehand or can be calculated. This is because, consecutive events will be destined to different outputs in a predetermined way, so that the load of the EFU can be balanced.

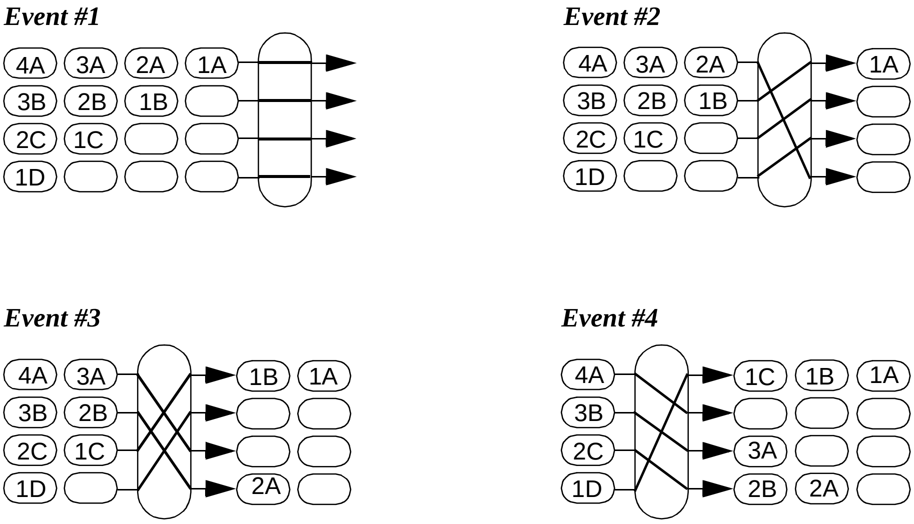

Figure 2.3: Barrel shift network

All the fragments of an event will always be addressed to the same destination. Next events, will be forwarded to different destinations, depending on the way the EFU scheduling is done. This rather simple traffic pattern may lead to serious competition for the same output of the switch, commonly referred to as output blocking. Because of this high traffic predictability, techniques like *barrel shifting* or more generally, *input traffic shaping* [Bars90] can be implemented to minimize the output-port blocking.

Figure 2.3 illustrates the operation of a simple barrel shift network. Each of the sources transmits event fragments to different destinations in each time slot. After a full cycle of input-output interconnections has taken place, all the fragments of the first event will have arrived to a destination and so on. The barrel shift network of Figure 2.3 assumes event fragments of equal size. Its operation can be further extended to the more general case when the event fragments are of non-equal size. The barrel shift network requires an external control to synchronize the transmission of the sources (or the connection configuration of the switch) in isochronous intervals. Building such a control for a switch with a relatively high number of ports can be a difficult task and be a potential bottleneck in the overall event builder performance. The barrel shift networks have been successfully deployed in event building systems with a relatively small number of switch ports.

Another possible way to reduce the output blocking is traffic shaping of the sources. The RDPM are designed to have the capability to randomize the order of injecting event fragments to the EVB network, relatively to the arrival order of requests from the EVM. This can be achieved by some logic built into the RDPM that will implement a randomization algorithm to selectively pick entries from the FRB and transmit the appropriate fragment.

Common to telecommunications and data networking, the event builder sources can implement the rate division technique, for traffic shaping. Using rate division, each source will transmit to a destination only a fraction of the link bandwidth, so that the sum of the individual rates reaching a destination is less or equal to the full link bandwidth. Rate division is mostly applicable to ATM networks, were each virtual connection can have its own specified sending rate. A survey of representative ATM switching techniques, is done in [Pryc94]. Recent work with 4×4 and 8×8 ATM event builders for the CDF experiment, has shown superior performance when rate division is implemented [Baue96].

The approach of rate division is particularly interesting, when parallel event building is adopted—the destinations assemble several events simultaneously—because the unused bandwidth during a fragment transmission is used for transmitting fragments to other destinations.

To construct a switch network with the number of ports envisaged in the CMS DAQ system, smaller switches can be connected together. The switches comprising the network can be arranged in a single or in multiple stages. A simple interconnection topology with a single stage is that of the cross-bar switch.

One major issue when smaller switches are cascaded together to form a larger network is the potential internal blocking that may occur under certain load conditions. This can happen when two different sources send to two different (or the same) destinations, but the messages must cross the same internal node of the network. The cross-bar switch offers several paths between two different end-points, hence internally has non-blocking properties. Output port contention can still occur, in particular when they are used for event building without any traffic shaping.

The cross-bar network fully interconnects N sources to N destinations and therefore it scales as $O(N^2)$ when new nodes are added. This scaling property of the cross-bar network makes it impractical for constructing large scale networks.

Better scaling behavior can be achieved with multi-stage interconnection networks (MIN). In a MIN, the path from a source to a destination can traverse one or several intermediate switches, assembled into stages. For that reason, a MIN can be more prone to internal blocking than a cross-bar switch. MIN interconnection topologies and their properties have been investigated since very early for telecommunications and data networking purposes [Ahma89]. Their blocking properties and scaling characteristics are very well understood today [Fahm95] [Turn97].

The main advantage of MIN that makes them more attractive than cross-bars, despite their inherent complexity, is their scaling behavior. They can be constructed with less crossing points than the cross-bar network and provide essentially the same non-blocking characteristics. The Clos network illustrated in Figure 2.4, has a third stage that provides at least one more path between any source and destination. A three-stage, N-port Clos network can be built by smaller switching elements of size $n \times m$ for the first switch stage, $p \times p$ for the second stage and $m \times n$ for the third stage, where the values for p, m and n are given as $p = \frac{N}{n}$, $m = 2n\text{-}1$ and $n \approx \sqrt{\frac{N}{2}}$

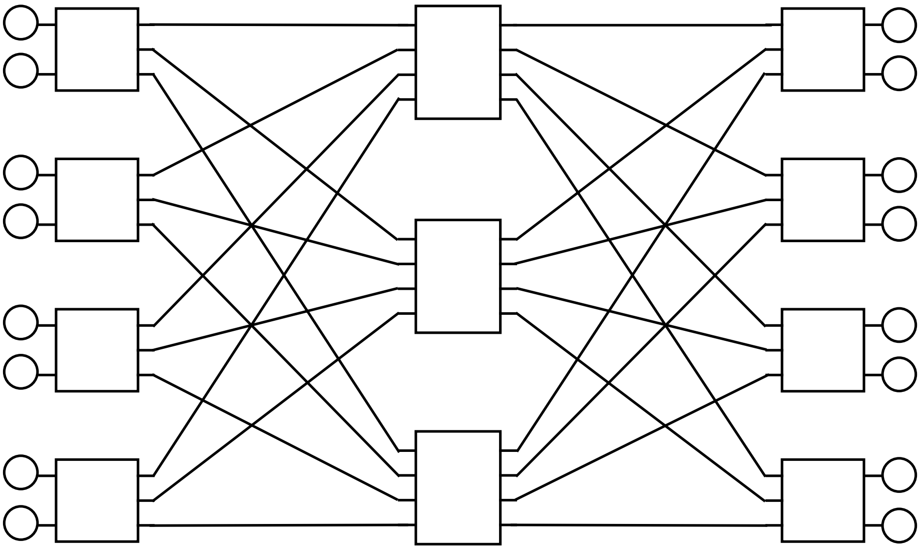

Figure 2.4: Clos network with 3 stages

The Clos network scales as $O(N\sqrt{N})$, hence for large number of input–output ports *N*, the total number of crossing points can be several times smaller than in a cross-bar switch. From the connectivity point of view, it can be seen that when $m > 2n - 1$ the Clos network can be internally non-blocking. However, it is much more difficult to obtain a similar non-blocking condition in cases like the ATM networks, where the individual virtual channels may vary their bandwidth requirements with time. More relaxed non-blocking conditions can be obtained by limiting the blocking probability to a sufficiently low value. Those types of networks are referred to as rearrangeably non-blocking.

Other types of MIN like Banyan [Goke73] and Benes [Turn97] have been investigated. Although they are internally blocking, they have a considerably better scaling behavior *e.g.,* $O(N \log N)$. An enormous amount of research work in the MIN area, has provided a lot of insight in their blocking characteristics. Under certain assumptions on the input traffic and the switch organization, Benes networks can have very low congestion probability [Bian93]. There is evidence that this non-blocking behavior can be expected also when the switch size reaches the size of the switch needed by the CMS DAQ system [Chan96].

Inherently blocking networks can be made non-blocking with appropriate time-ordering of the data entering the network. This is more difficult to achieve in a telecommunications or data network, were the traffic is assumed to be of random nature. It is however much easier in the event building case where the connection patterns are well defined.

The fault tolerance of the switch fabric is another important issue in the event builder design. Depending on the switch architecture, a single link failure in the switch may disable several destinations in the EFF. A switch with a large number of ports, as it is considered here, can be particularly prone to failures. Careful selection of the switch interconnection topology and its technology can minimize the effects of possible failures.

2.1.3 The Switch-to-Farm Interface (SFI)

The SFI is the connection point between the EVB switch fabric and the EFU. The last stage of the event building is carried out by the SFI. Each switch output has its own SFI, which is attached to one EFU. Each SFI has to assemble the incoming fragments belonging to the same event into separate entities of sub-events (for the LV2-trigger) or full events (for the LV3-trigger).

The SFI operation has many similarities with that of the RDPM. Its function can be conceived as that of an inverted RDPM. It has a network interface to connect to the switch fabric and an I/O bus to connect to the EFU. Its internals are essentially the same with the RDPM, with the main difference that its output is not driven by the EVM. Instead, it is driven from the EFU that sends requests for the next assembled event to be processed.

The current design of the CMS DAQ architecture, foresees a slightly modified version of the RDPM to be used for the SFI. It will be enhanced with the necessary logic to communicate to the EVM the EFU requests for reading a new event, clearing a processed event or discarding a LV3-part of an event.

Alternative ways to implement the SFI functionality are considered also. One is the use of a modified version of the DSP-based RDPM (VORTEX) [Bran95]. This alternative is attractive mostly because of the simplicity and versatility of the VORTEX design, that makes it more easy to adapt from operating as RDPM, to the SFI operation. It also requires additional logic to interface to the EVM network.

A second alternative is to emulate the operation of the SFI, in software that is running on the EFU. This idea of an emulated SFI is particularly attractive, because it eliminates any hardware design, construction and maintenance as required for a custom designed system. An emulated SFI is much simpler to build, provided that the EFU that can have the required additional resources to assemble the incoming event-fragments into events. As the previous SFI candidates, the interface to the EVM is not addressed either. It can be thought however that this task can be easily handled from the EFU itself because very few additional resources will be required. The software emulated alternative is extensively studied in Chapter 6.

The SFI has to be capable to handle the incoming event-fragment rate, resulting from the requests of its attached EFU. The same requirement is valid also for the network interface controller that connects to the switch fabric. Estimated values of the incoming event rate are approximately 100 new events per second for a farm of 1,000 EFU. The arrival rate of individual fragments however, is much higher. The bursty nature of the incoming traffic of the SFI, can have severe implications on the SFI performance. The effects of this on the network interface controller will be studied in Chapter 6, together with the implications on the overall SFI performance.

2.1.4 Event Manager (EVM)

The event manager is the key component for the synchronization of the event builder. It is a custom made hardware device that has the necessary logic to associate each LV1-trigger event to a destination in the EFF.

The EVM is notified by the different EFU for their availability, by messages containing filter resources (FRS). Each FRS, points to an available buffer in one EFU. It is expected that the EFU independently from the occurrence of the LV1 triggers, will send to the EVM such messages. The EVM will hold the FRS in a queue or a table. At a LV1-trigger, occurrence, the EVM will pick an FRS, it will assign to it an event number and broadcast it to the LV2 or the LV3 group of RDPM. Another function of the EVM is to forward to the LV3 group of RDPM, the CLEAR messages for the sub-events that failed to pass the LV2 trigger.

The functionality of the EVM is rather simple. However, in order to minimize the event building latency, the requirements on its response time are quite high. Ideally the EVM could provide an FRS to the RDPM in a time interval equal to the FED read-out latency of the RDPM. The EVM is also the place where the EFF scheduling can be done. With some intelligence added to the EVM, it can order the broadcasts of FRS according to a given scheduling policy. Such scheduling policy could be based for instance on the frequency of arrivals of the FRS from each of the EFU. This can be particularly useful in the case of a heterogeneous EFF *i.e.,* not all the EFU have the same capacity.

The communication media used between the EVM with the RDPM and between the EVM and the EFU, is an important design issue. Between the EVM and the RDPM, a multicast type of communications seems appropriate. It has relatively modest bandwidth requirements but the type of the expected traffic will be very bursty. More complicated is the case of the network utilized between the EFU and the EVM. It will also have modest bandwidth requirements, but it may be severely congested due to the high number of sources sending to a single destination.

Equally important to the functional design of the EVM, is its tolerance to failures. The EVM, in the current event building architecture has a prominent function from the reliability point of view. Failures of the EVM, will prohibit any event building to take place. A redundant design seems appropriate to reduce to a minimum the time that the DAQ system is unavailable due to EVM failures.

2.2 The Event Filter Farm (EFF)

The concept of using a farm of computer systems for the event selection and processing, is a common practice in HEP experiments. With the advent of powerful workstations and personal computers, farms of commodity equipment have become very attractive solutions for the on-line purposes of HEP experiments like in HERA-B [Gell95]. The main reason for that is to provide enough capacity to cope with the LV1-trigger rates. The different events accepted from the LV1-trigger, are not correlated by each other. Hence, the idea of filtering events into several independent nodes, that do not require any kind of interaction seems appropriate. To satisfy the physics requirements of the LHC experiments, a high LV1-trigger rate must be accommodated. The CMS DAQ system is designed to be able to operate at a maximum sustained LV1-trigger rate of 100,000 events per second (Table 1.5).

2.2.1 Architecture

The CMS EFF, consists of some 1,000 EFU connected to the EVB switch through the SFI. It must provide a sufficient amount of processing power to accommodate the maximum LV1-trigger rate and the load of the LV2 and LV3 filtering processes. Such a design of the event filtering part of the DAQ system, has the advantage of a very flexible scaling behavior. Depending on the processing power of the EFU deployed in the EFF, their number can be reduced or increased. Also, due to the nature of the event filtering —non-related events— multiprocessor systems can be easily assumed. Again the aggregate computing power required by the EFF can be achieved by adding more processing units in the EFU, given sufficient expansion capability.

The current construction schedule of LHC, envisages a period of a few years of operation at lower luminosity. During that period, the expected physics event rate will be much lower than that of nominal luminosity operation. A flexible EFF, can be initially built with either less powerful EFU or with a smaller number of EFU that will be sufficient for the low-luminosity period and later can be upgraded to its nominal power.

The ability to design a flexible event filtering system is of utmost importance for the success of the CMS DAQ system. With a relatively long design and construction phase until the year 2005, and the immense progress of the computer industry, it is very difficult to make precise predictions of the technology and the computing power that will be economically available at that time. Additionally, unexpected physics phenomena may need to be studied, resulting in event rates higher than what the current physics simulations suggest.

2.2.2 Scheduling

A farm with the size needed by the CMS DAQ architecture, must also employ a flexible scheduling of its resources. A simple approach could be the EVM to allocate EFU to new LV2 and LV3 triggers in a round-robin manner. However, a more sophisticated farm scheduling seems to have some obvious advantages. Firstly, EFU fault isolation will be straightforward. The faulty EFU can be removed immediately from the list of available destinations, thus preventing events to be forwarded to a non-functioning EFU. Secondly, during the lifetime of the CMS experiment, several upgrades of the EFU may take place. This will result into a non-homogeneous EFF consisting of EFU with very different performance and capacity. A dynamically scheduled farm, can better balance the event filtering load amongst different EFU. A possible way to implement dynamic EFU scheduling could be based on monitoring the rate of requests arriving from each of the EFU. The EFF scheduling policies can be implemented by the EVM.

2.2.3 Control and Management

Besides the main control exercised by the EVM for scheduling purposes of the EFF, functions like EFU initialization and restart, status monitoring and alarm condition detection must also be carried out. This type of control, will be part of the general detector control system (DCS) that will command all crucial DAQ sub-systems.

Alarm conditions and EFU failures together with diagnostics information will be communicated to the operators, so that the necessary actions can be taken. Data collected from the EFU status monitors, will be analyzed continuously to ensure the correct EFF operation.

The initialization of the EFF and management of its resources into smaller partitions is also relevant. This can be particularly useful for testing new event selection algorithms, or during periods of lower luminosity, some of the EFF resources can be allocated for tasks related to the off-line data analysis.

2.2.4 Alternative Solutions

There are also several alternative ways to build the EFF. They can be more generally classified into custom and commercial alternatives. For the purposes of the CMS DAQ system, building a custom designed EFF may have a tremendous financial impact. The design, construction and maintenance costs of an EFF consisting of custom made EFU (*e.g.*, based on DSP) may quickly become dominant to any advantages gained by such a design.

From the commercially available alternatives, the massive parallel processing (MPP) systems seem very close to the EFF and EVB architectures. The high speed communication network, the large amount of computing power and sometimes the scaling flexibility are their more prominent resemblances. The MPP systems are not considered because most of them are proprietary products and their integration (even though sometimes it is based on commercially available systems) follows a different and slower path than that of the widely commercialized systems. For the lifetime of the CMS experiment, the MPP market is considered too small to offer economical solutions and upgrades.

Another possible alternative could be the use of embedded systems, adapted to the EFF requirements. For instance, a farm consisting of digital signal processors could be constructed to offer the required performance. This possibility is not considered for the same arguments of wide commercial availability and proprietary origins. In addition, the adoption of an on-line computing system used for event filtering that is entirely different from that used for the off-line analysis, is not very attractive as it will make more difficult the development of the event filtering algorithms.

2.3 The Event Filter Unit (EFU)

The EFU is a computer system, where assembled sub-events or full events will be processed by the LV2 and LV3 filtering processes, respectively. It has to provide sufficient computing power to ensure that the rate of filtering events is higher than the rate of event arrivals. Also it must have the necessary memory and I/O bandwidth to handle the incoming (and to a less extent the outgoing) event data-stream and also the needs of the filtering processes themselves.

The EFU interfaces to the EVB network through the SFI. It also has a connection to the computing services network, which will collect the events accepted by LV2 and LV3 triggers for permanent storage. A service network for monitoring and control will also connect to the EFU.

2.3.1 Architecture

The main requirement of the EFU is to provide the necessary computing power needed by the LV2 and LV3 filters. At the current early design phase of the experiment it is difficult to quantify this requirement. Rough estimates based on extrapolations of similar physics experiments, indicate that some five million MIPS in total will be required by the filter algorithms. This estimate is well ahead of what current processor performance can offer to build a farm, but it is still in the limits of the processor extrapolated performance by the time when the CMS experiment will start.

In the next years to come until the start-up of the experiment, the computer systems architecture is believed to undergo significant changes —apart from improving processor performance. The potential of such changes is to overcome today’s difficulties as is the main memory performance. In this context, the advantages of using off-the-shelf commercially available systems for the EFU, are very important. It is also the most economic way to fulfill the DAQ requirements. Today, a trivial and economic way to increase the capacity of a computer system is to add more processors to it. It is believed that this approach will continue to apply for future computer systems.

Given the flexibility of the EFF to adapt to a wide range of processor performance evolution scenarios, we suggest that it is appropriate to use multiprocessor computer (MP) architectures for the EFU. Our main assumption here is that the DAQ performance will profit by the increased processing capacity. That will avoid more expensive solutions as adding more switch output ports and EFU to the system. The non-parallel and compute intensive nature of the filtering software as it is conceived today, relaxes the scaling requirements of MP systems, compared to the case of highly parallel filtering algorithms.

2.3.2 I/O Interfaces

During the EFU operation, significant amounts of data will need to be transferred to the EFU despite of the numerous EFU. For example, let us consider a system that has to sustain a LV1 trigger rate of 100 kHz and consists of 1,000 EFU, as was defined in Table 1.5. If we divide the LV1 trigger rate with the number of the EFU, we arrive at the average input rate of LV2 sub-events in the EFU, that is 100 LV2 sub-events per second on average will be transferred to each of the EFU. This results to a sustained I/O bandwidth of approximately 30 MB/s . This figure is a function of the LV2 sub-event size, the number of the EFU in the EFF and the number of events accepted by the LV2-trigger in a unit of time.

An EFU that is based on a commercial computer system with a full-fledged operating system may have difficulties to ensure that the needed bandwidth is delivered to the filtering applications. It is worth noting that independent of the SFI implementation (software or hardware) the I/O subsystem of the EFU is in a very critical performance path and must be carefully studied. This problem of interfacing to the operating system, together with its implications is investigated in detail in Chapter 6.

The I/O interface of the EFU with the computing services network, has modest requirements of approximately 100 KB/s *i.e.,* one full event to be accepted every ten seconds. Very little bandwidth is required by the service network used for the control and monitoring of the EFU.

2.4 Filtering Software

The LV2 and LV3 trigger algorithms are the last two stages of the on-line processing of the events recorded by the CMS detector. They must be able to efficiently distinguish the interesting physics events, in a short time interval. The filtering software is tightly coupled to the software that will be used for the full event analysis, at the off-line processing stage. It is required therefore, that a similar to the off-line software approach is used also for the on-line. Modern techniques of software engineering need to be adopted to guarantee the quality of the filtering software. Therefore, the presence of a software environment similar to the off-line software is important for the on-line. This will significantly ease the development, maintenance and improvement of the filtering software.

In order to quantify what is the computing power needed in an EFU, we need to be able to characterize and quantify what will be the workload of the LV2 and LV3 trigger algorithms. This is particularly difficult in the current early design phase of the high-level triggers, as most of the software components related to it are currently under design. Experience from previous similar experiments indicates that it is safe to assume that the filtering software is mostly CPU bound. It demonstrates significant locality of references to main memory and in most of the cases can fit into today’s processor caches.

For the purposes of this work, it seems that is only possible to know an estimate of the needed time by the LV2 and LV3 trigger software. The average execution time of a high-level trigger will be used instead, which essentially describes the ratio of the processing path length over the processor performance.

An important characteristic of the high level triggers is their rejection ability of the events they process. This property is entirely determined by the behavior of the trigger algorithm. The rejection ability of the LV2-trigger is of crucial importance, because it determines the incurring rate of the LV3-triggers. The ratio of the LV1-trigger rate over the resulting rate of the accepted events by LV2-trigger, is called the *LV2-trigger rejection factor* —*R*. A similar characteristic can be formulated for the LV3-trigger algorithm.

3 System Architectures

In this chapter, we introduce the basic building blocks and operation principles of the MP computer systems.

Multiprocessor (MP) computer systems make their appearance in the 1970s. Most of them were research projects, like the pioneering *c.mmp* [Mash82] and *cm** projects at Carnegie–Mellon University, while very few were successful enough to be commercialized. Commercial MP systems start to become available in the mid 1980s. As the dramatic evolution of microprocessors made available computing power in affordable prices and sizes, a clear trend for MP systems is created in the 1990s.

3.1 Multiprocessor Architectures

A very popular classification of MP systems according to Flynn [Flyn66] can be done based on the number of data and instruction streams they can process simultaneously:

- **S**ingle **I**nstruction, **S**ingle **D**ata streams (SISD)
- **S**ingle **I**nstruction, **M**ultiple **D**ata streams (SIMD)
- **M**ultiple **I**nstruction, **S**ingle **D**ata streams (MISD)
- **M**ultiple **I**nstruction, **M**ultiple **D**ata streams (MIMD)

In the first category, SISD, fall the uniprocessor (UP) systems. In the MISD category, only a few experimental systems have been implemented, but none of them did appear as a commercial product.

In a SIMD architecture environment, the processors are synchronized from a global clock and execute the same instructions on different data streams. SIMD architectures are known to perform very well for a rather specialized subset of applications with repeated calculations over large array data structures.

Systems with a MIMD architecture consist of nodes with UP or MP systems connected to a high speed communication network. Some of these nodes act as disk, or more general as I/O servers. The interconnection network consists of high performance and scalable switching systems, often offering the possibility of including additional nodes to the system.

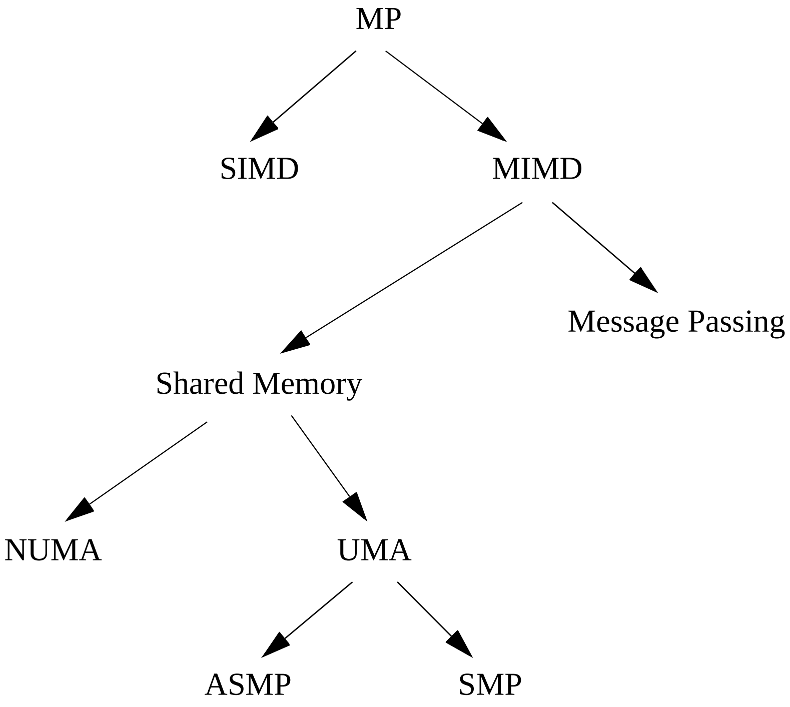

Figure 3.1: Multiprocessor classification

As depicted in Figure 3.1, further classification of MIMD architectures to *shared memory* and *message passing* systems can be done, based on the way data can be shared. Shared memory systems make available the memory resources to all nodes, no matter if the memory is local to a node or not. When the memory is directly connected to the communication network of a shared memory system, these systems are called uniform memory access time (UMA) systems, while systems having the memory modules distributed to each node are called non-uniform memory access time (NUMA) systems. In message passing systems, memory is not physically shared, instead, short messages are exchanged between the nodes through the communication system, containing either data or synchronization primitives.

Systems in the general MIMD category are often referred to as *loosely coupled* systems as every node is a rather autonomous entity with its own memory and I/O channels. Their counterparts, *tightly coupled* systems, consist of shared memory systems with processors sharing the main memory and the I/O devices. Examples of systems falling into the most common categories are summarized in Table 3.1.

System	**NUMA**	**UMA**	**Message Passing**	**Shared Memory**
Intel Paragon			•	
Stanford DASH	•			•
SUN UE-6000		•		
Thinking Machines CM-5			•	

Table 3.1: MP Systems

3.1.1 Symmetric Multiprocessors

Symmetric multiprocessor (SMP) systems are basically tightly coupled shared memory systems. As depicted in Figure 3.2, typical SMP systems have processors, memory modules and I/O devices connected to a common high performance bus system. They are *symmetric* in the sense that processors and I/O devices have equal access to memory. Additionally, any processor can access independently the I/O devices. There are no privileged processors on the system for any operations, like access to the I/O sub-system or execution of the operating system code.

In contrast to the SMP systems, there is a smaller category of asymmetric MP systems (ASMP), where one of the system's processors handles operations like access to the I/O devices. They are less popular than the SMP systems, due to the obviously less performance they can offer due to the bottleneck of the privileged processor. However, they are simpler to design.

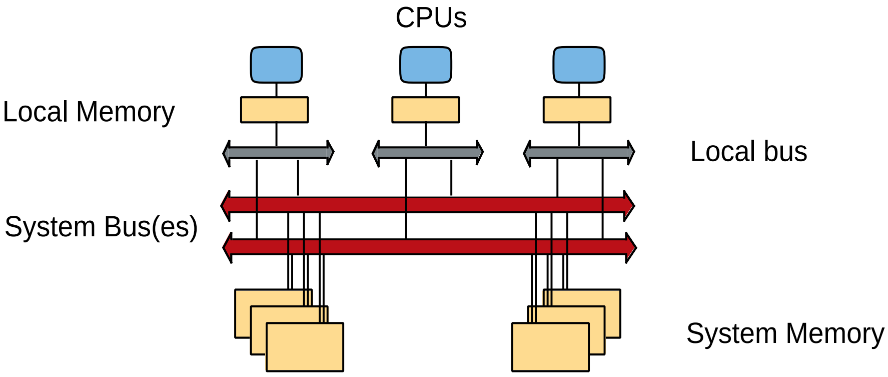

Figure 3.2: SMP Architecture

SMP systems are a recent and a rather successful addition to the high-performance computer marketplace. There are many reasons for that, amongst them:

- Utilization of traditional uniprocessor components
- Small scale systems with very high computing capacity
- Possibility to rightsize the system to the application, by adding more processor and memory modules.

One important parameter of an SMP system is *scalability*, *i.e.*, the ability to increase the total performance of the system by adding more processors. Every system has an upper limit of scalability, which usually depends on the characteristics of the interconnection (bandwidth, arbitration policy, latency, etc.), the memory bandwidth, the I/O bandwidth, and the applications running on the system. More sophisticated systems, come with multiple buses and few of them, with crossbar interconnects in order to increase scalability.

In an SMP system different applications can run on different processors, without interference between them. However, applications may share the same memory. Different synchronization techniques are offered by the operating systems, to ensure data integrity. At the hardware level though, if more than one processor reference the same memory location, then arbitration will be done at the bus level to serialize the memory request. In conjunction with the system's memory model, the arbitration techniques could be quite complex.

On SMP systems, applications do not necessarily run faster. As we saw above, each processor may execute different applications, hence an SMP will offer at least more computing capacity. However, it is possible for specially built applications to run on more than one processor, resulting in lower total execution time. It has to be noted that even in that case processor instructions will be executed at the same speed.

3.2 Microprocessors

The microprocessor technology is probably the fastest evolving one. It is estimated [Henn90] [Henn91] that microprocessor speed increases by 50% to 100% every year. This increase rate is mostly attributed to the developments in architecture design and to the dramatic increase of semiconductor integration in VLSI chips. Table 3.2 shows some of the microprocessors of the last decade and the evolution of their clock speed and architectural characteristics.

Modern microprocessors adopt techniques like caching, pipelining, branch prediction, as well as multiple functional units to increase performance. Using pipelines, execution of instructions can be overlapped. Processors with more than one pipeline are called *superscalar*. The depth of the pipeline also has permitted significant reduction of the cycle time of processors. Processors incorporating pipelines deeper than 5 stages, are called *superpipelined*.

Processor	**Year**	**Type**	**Clock (MHz)**	**Issue**	**Stages**	**Units**	**On-chip Cache (i-d) KB**	**Silicon (μm)**
i486	*89*	*CISC*	*33*	*1*	*5*	*2*	*8*	*1.0*
68040	*89*	*CISC*	*25*	*1*	*6*	*2*	*4-4*	*0.8*
Micro-SPARC	*91*	*RISC*	*50*	*1*	*?*	*1*	*4 - 2*	*0.8*
Super-SPARC	*92*	*RISC*	*60*	*3*	*4 - 5*	*5*	*20 - 16*	*0.6*
Alpha 21064	*92*	*RISC*	*200*	*2*	*7 - 10*	*4*	*8 - 8*	*0.75*
MIPS R4000	*92*	*RISC*	*100*	*1*	*8 - 10*	*2*	*8 - 8*	*1.0*
Intel P5	*93*	*CISC*	*99*	*2*	*5 - 8*	*3*	*8 - 8*	*0.8*
HPPA 7200	*94*	*RISC*	*120*	*2*	*5 - 6*	*3*	*? - 2*	*0.55*
SPARC Ultra I+	*95*	*RISC*	*200*	*4*	*9*	*9*	*16 - 16*	*0.42*
Power PC 601	*95*	*RISC*	*66*	*2*	*4 - 6*	*4*	*4 - 4*	*0.5*
MIPS R10000	*95*	*RISC*	*200*	*5*	*5 - 7*	*5*	*32 - 32*	*0.35*
HPPA 8000	*95*	*RISC*	*180*	*4*	*7 - 9*	*10*	*?*	*0.5*
Power PC 604e	*96*	*RISC*	*225*	*4*	*4 - 6*	*6*	*32 - 32*	*0.35*
Alpha 21164	*96*	*RISC*	*500*	*4*	*7 - 9*	*4*	*8 - 8* *+ 96KB L2*	*0.35*

Table 3.2: Processor architecture evolution

Branch prediction algorithms were implemented to predict the resulting address of conditional branches met in programs' code. If a branch is correctly predicted, the pipeline will continue to work without disruption. Otherwise, the pipeline will have to be flushed and many processors cycles spent until is filled again. The algorithm of branch prediction maintains historical records of the code branches into special buffers built into the processor's chip.

Almost all processors today incorporate cache memories in their chips, in order to keep to a minimum the main memory access latency. Some of them like the Alpha 21164 also have a secondary level cache on-chip.

Further increase of the speed of microprocessors was achieved with the appearance of RISC (Reduced Instruction Set Computer) processors. In contrast to their CISC (Complex Instruction Set Computers) counterparts, lesser and simpler processor instructions were used in order to get close to the desired target of one cycle per instruction. However, the elimination of the more powerful and cycle-consuming instructions of the CISC generation, resulted into longer program codes.

RISC processors today are highly pipelined implementations, as this is the easiest way to achieve the goal of one instruction per cycle. As Stone [Ston87] notes, chip area can be used instead to decrease instruction and data traffic. An example of such implementation is the SPARC processor [SPAR90], a spin-off from the Berkeley RISC II project [Patt82], where registers are grouped in so-called *register windows*. Programs running on such processors are using overlapping register windows to pass arguments between procedures.

The dramatic improvement of microprocessors, led to workstation systems with computing power equivalent to that of mainframes and the replacement of supercomputers by massive parallel processors (MPP). As examples of future directions in processor architecture, one could mention multi threaded architectures as well as the more specialized very large instruction width (VLIW) processors and digital signal processors.

3.3 Memories

Computer systems incorporate different kinds of memory modules, usually distinguished by their capacity, access time (latency) and cycle time. The latency and the cycle time (which fixes the data transfer speed of a memory module), are important performance characteristics. The technology used for their constructions usually determines the above characteristics.

3.3.1 Virtual Memory

The memory size of a computer system is a very important resource and design parameter. Already from the UP system generation, techniques like virtual memory were introduced to give the illusion of almost infinite physical memory to application programs. Additional hardware, the memory management unit (MMU), was added to systems in order to make more efficient the memory management. Its task is to map dynamically the physical memory addresses to virtual memory ones.

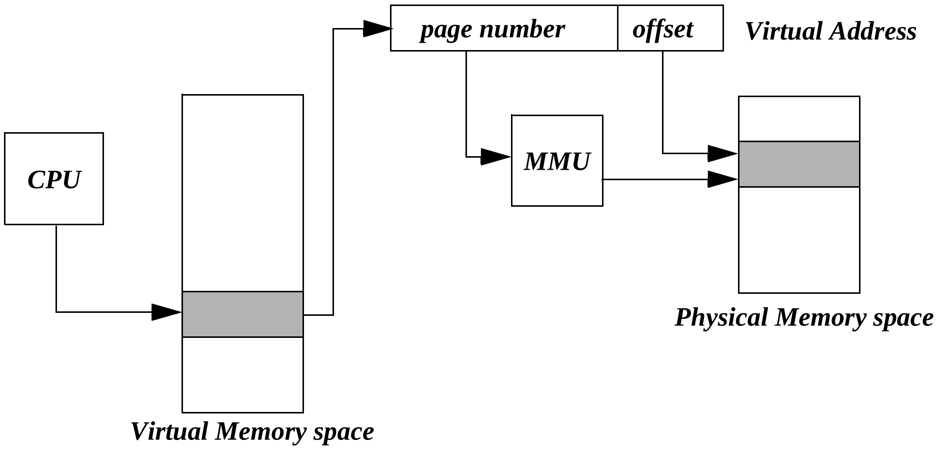

Figure 3.3: Virtual Memory layout

The physical memory is split into equally sized segments called *pages* and one translation is done for each page. Figure 3.3 shows the way this translation is done and how individual memory locations are accessed.

During the execution time of a program, virtual addresses are referenced. When a virtual address with no physical mapping is referenced, a *page fault* occurs. This results to a processor trap, which is handled by the operating system. An attempt to find a free page from the physical address space will be made. If there is no free physical page, a used one will be replaced and its old contents, if modified will be stored to disk, otherwise will be discarded. At that stage, the memory translation can be established.

The virtual memory mechanism is handled by the OS, in order to make more efficient utilization of the memory resources between different applications. Some pages can be locked in physical memory permanently, avoiding the overheads of fetching them from the disk. This is particularly useful for high performance applications and it is always used by the kernel of the OS. A very important performance factor is the *page replacement policy* adopted by the OS. Many algorithms are implementing different page replacement policies. The least recent used (LRU) policy is one of the most commonly met.

The MMU design determines important system parameters as the access protection mechanism of memory pages, the page size, as well as the size of the virtual and physical address spaces. The choice of the page size is an important trade-off between overheads and management of the memory resources. Common page sizes are 4 KB, while some systems like the UltraSparc have page sizes of 8 KB. Systems built from modern processors of the ALPHA, MIPS and SPARC families, may have variable page size for better management of large code segments.

The address translation time can be further reduced if the MMU caches the address translations. This cache is called *translation look-aside buffer* (TLB) and has essentially the same structure as cache memories.

3.3.2 Memory Hierarchies

During the last decades, the main memory of computer systems evolved from a single module to a multi-level hierarchy of memory modules with different characteristics. The reason for that is to optimize the needed capacity, the data transfer speed, the access time, as well as the cost of the system and match the speed of the processor modules. Less performing memory devices *e.g.,* disks, are cheaper and can be afforded in larger quantities, while faster and high performance devices are much more expensive and can be afforded for few only parts of a memory hierarchy.

On the other hand, the evolution of microprocessor technology produced processors far much faster than common memory modules could stand. In any system, there is very little to be improved by just increasing the speed of the processor without at the same time keeping in balance the memory speed.

The size of each of the building blocks of a memory hierarchy are important parameters in the design of a system and play an important role in the overall performance. A typical memory hierarchy is depicted in Figure 3.4. As a general rule, the higher the level in a memory hierarchy is, the faster and smaller sizes of memory modules are used. At the lowest level of the memory hierarchy are placed devices with the additional property of being capable to keep data even across power-off periods.

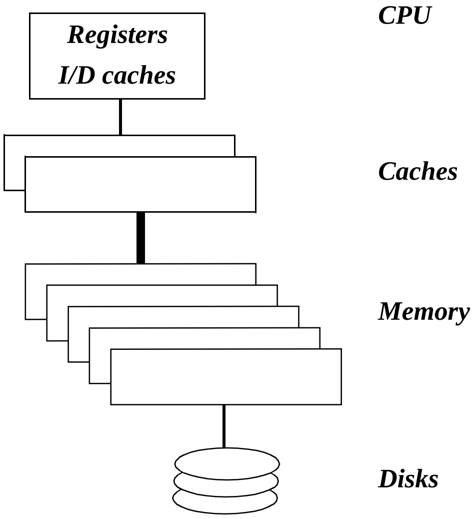

Figure 3.4: Memory Hierarchy

This model works well due to the principle of *locality of reference*, that is, at any given time of a program's execution only small segments of its address space are accessed. By keeping any time during program's execution short segments of its code in faster memory modules (and thus hiding the time consuming access to slower memory modules), the execution speed increases significantly. One may distinguish two kinds of locality of reference; the spatial and the temporal. In *spatial locality of reference*, adjacent memory segments are likely to be accessed soon, while in *temporal locality of reference*, accessed memory locations are very likely to be accessed again soon.

Starting from the top level of the memory hierarchy, the processor's registers offer the fastest (1 clock cycle) but smallest (few hundreds of bytes) storage space. Some of the processor registers are used to store operating system data during the execution of a program, while most of the registers have the property of direct arithmetic manipulations of their contents.

The primary cache memory is the next level. Some systems can have an additional cache memory level, the secondary cache. The primary and secondary cache memories, are often referred to as Level-1 and Level-2 cache, respectively. Typically, the primary cache is in processor's chip, while the secondary is not. Primary caches come with sizes of few tens of KB, while secondary caches can reach even 4 MB. Modern designs incorporate split primary caches for the processor's instruction and data streams. This is due to the different locality of reference behavior for code and data segments.

Next to the cache memory is the main memory system which can have any size below a limit, determined by the design of the system and limited by the width of the address bus of the processor. The access time to the main memory can reach even few hundreds of clock cycles depending on the operation that it is been executed. At the bottom of the hierarchy, are the storage devices, usually magnetic disks, with capacity reaching few GB and access times usually below 10 ms. The transfer speed of disks to main memory is substantially lower than that achieved between other levels of the hierarchy, with typical values below 50 MB/s.

3.3.3 Cache Memory Systems

As was described above, the cache memory is called to abridge the performance gap between the main memory and the processor. The management of the cache memory is transparent to the user application and it is handled by the hardware or the operating system itself. That way, portability between systems with different cache organizations can be ensured.

The principle of cache operation can be summarized as follows. The portions of main memory referenced by a program at a given time are stored into cache locations called cache lines. The identification of the memory portions which are resident into the cache is done by tagging the data (or instructions in an instruction cache) with its address in the main memory. When the processor references a main memory address, this address is forwarded to the cache where the hardware makes a search of the cache tags and determines if the data is resident or not. If the data is found, then we have a *cache hit*, otherwise a *cache miss*. In the case of a hit, the data is passed back to the processor, otherwise a bus request is started to fetch the data from its main memory location. In the last case, a copy of the main memory location will be stored into the cache to take advantage of the temporal locality. Systems with secondary cache memory, operate in a similar manner.

The cache entry replacement policies are simpler than those of the virtual memory systems. Usual algorithms are LRU, as well as random replacement.

Different techniques are used to search the contents of a cache memory. The most popular one is hashing the addresses and searching the cache with the hash index. Cache memory contents can be organized in different ways. Some of them are:

One-way set associative. The correspondence of memory locations and cache memory entries is one-to-one. They are also called *direct mapped* cache memories. They are subject to *thrashing i.e.,* cache lines are continuously replaced before any hit occurs.

Two-way set associative. The hashing algorithm will index a set of two lines in the cache. Their main advantage is that cache thrashing can be reduced. They are more complex to build.

Fully associative. Any cache line can be accessed, hence no hashing algorithm is used for indexing. They minimize the cache thrashing, while are more expensive to build. They are used for rather specialized cache memories like the TLB cache of the MMU.

Cache memories can be accessed either by virtual or physical addresses. The virtually indexed cache memories are faster to access, as no MMU translation is required. Their entries are subject to *ambiguities i.e.,* the same virtual address can be used by different processes, thus requiring often flushing from the OS. Physically indexed cache memories instead, are no subject to ambiguities, need less OS support, but require MMU translations on every access.

The policy of updating the cache contents (also known as write policy) influences the behavior and performance of the cache. The write policy determines how data is stored into the cache and the main memory. There are two possible write policies. With *write -through* policy, once the processor writes into the cache, the new values will be transferred immediately to the main memory. Write-through has the obvious disadvantage of slowing down the cache accesses in order to keep the main memory consistent with the cache contents. With the alternate *write -back* policy, the new contents of the cache will not be forwarded to the main memory except if the corresponding cache line is replaced or the OS forces that. A special bit in the cache line called the *dirty bit* is used to keep track of modified entries. In contrast to the write-through policy, write-back leaves main memory inconsistent, requiring the frequent intervention of the OS. However, it speeds up the access to the cache because updates to the same cache entry will be forwarded only once to the main memory.

The current trend on multi-level cache systems is to have the primary cache virtually indexed directly mapped and write-through in order to increase the processor clock speed, while secondary caches are usually write-back.

3.3.4 Memory Models

The memory model of a computer system defines how memory is accessed during load and store operations. In particular for SMP systems, it also defines how are handled simultaneous accesses to memory.

The most common memory model is the *sequential model*. In that model, also called strong ordering, all program's instructions are executed in the order they appear in program's code. In SMP systems using that model, any load and store instructions are defined to be *atomic.* Atomicity means that operations executed by one processor are guaranteed to not interfere with those executed in other processors. This memory model is used by the majority of the systems.

Other memory models, mostly met in SPARC processors, are the total store ordering (TSO) and the partial store ordering models (PSO). Their main purpose is to increase processor's performance during write operations. For that, a store buffer (also called write behind buffer) is added to the processor, which actually represents an additional level of the memory hierarchy.

In the TSO model, data of store operations will not be forwarded to the cache memory immediately, but it will be placed into the store buffer. Load operations will check first the contents of the store buffer and if a hit occurs they will return the value found associated to the most recent store, otherwise the rest of the memory hierarchy is used. With TSO the contents of the store buffer are send to the cache memory in first-in first-out (FIFO) order.

The PSO model is similar to the TSO, except that only the stores to the same memory location in the store buffer will be carried out in FIFO order, leaving the order of other stores non-deterministic.

Non-strong ordering memory models allow processors to continue execution while slow memory updates are taking place. However, they are not transparent to all programs written for strong ordering models and additionally they have increased requirements on the way atomic operations in an SMP system are ensured.

3.4 System Networks

Several types of interconnection networks exist between MP nodes, as well as between processors, memory modules and I/O devices. Usually they are classified by their intrinsic topology. One can distinguish between buses, crossbars, fat trees, as well as rings, hypercubes etc. based systems. The choice of the type of interconnection is usually based on the architecture of the computer system they are deployed. For instance, in SIMD systems where point to point communication is essential, hypercubes are often met. In MIMD architectures the cache coherency requirement and the memory access methods, influence the choice of the interconnection. In NUMA systems such as the IBM SP-2 and Meiko CS-2, multistage networks are used. In shared memory systems, where cache coherency is essential, bus based interconnects are very common.

Computer systems often deploy more than one type of interconnect at different levels of their design. I/O devices are usually connected to the system through an I/O bus. The processor to memory interconnects are usually proprietary designs, while I/O buses tend to be more standard. Using standardized I/O buses, I/O interfaces can operate in different systems implementing the same I/O bus.

3.4.1 Bus Interconnections

Buses are used for processor to memory and I/O interconnections. They are traditionally classified [Henn90] as *processor to memory buses* (or just system buses) and *I/O buses*. Both consist of sets of electrical wires, except in the case of the *active bus*, where in addition to that, other electronic components are used to enhance their functionality. Buses have varying lengths, limited to few tens of centimeters. This limitation comes from the number of devices connected to the bus and from the speed at which signals can traverse it.

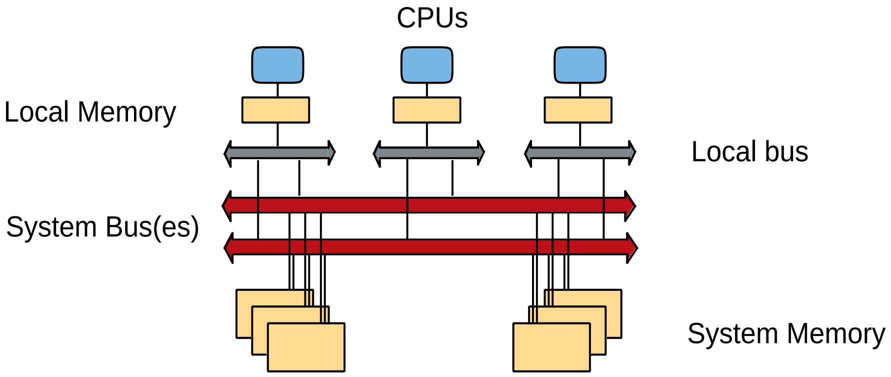

Figure 3.5: Bus based SMP system

Bus designs have to accommodate the speed of the processor and memories. Depending on the number of the devices connected to a bus and the electrical limitations, it is difficult to design buses with arbitrarily high speeds. Consequently, SMP systems utilizing buses as their interconnect may not easily scale, or in the worst case, may be limited in performance even with few processors only.

As depicted in Figure 3.5, SMP systems may have more than one bus connected in parallel. This is usually done to overcome the performance limitations of the single bus, but results in a more expensive design. An example of a twin bus based system is the SUN SPARC Server 2000.

I/O buses are usually connected to the system bus through a bridge device. Some systems may have the same system and I/O bus in order to reduce costs.

3.4.2 Cross-bar Interconnections

As was discussed in 3.4.1, the bus of an SMP system may easily become the performance bottleneck if the number of processors increases. For that, different interconnects have been designed. One of them is the cross-bar (Figure 3.6). In a cross-bar based system the interconnect resembles a grid, with processors, memories and I/O devices connected to its edge-nodes. Every cross-point has a switching circuit that is turned on, when a path between a memory and a processor has to be established. Often in systems with a cross-bar the memory is *interleaved*, that is, consecutive memory locations are stored in different memory modules. This is done to overlap the memory access time.

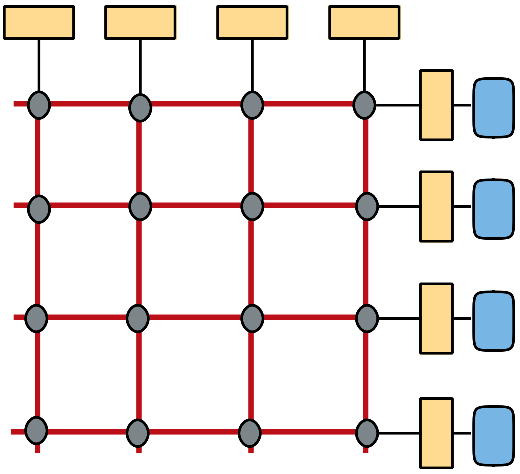

Figure 3.6: Cross-bar based SMP system

The cross-bar architecture qualifies as an SMP system, because memories, processors and I/O are tightly coupled and accesses to memory are still symmetric.

Cross-bar based systems have the advantage that accesses to memory and I/O can be done through many and independent paths, increasing the bandwidth and reducing the probability of contention. They offer higher system performance, than the bus based systems. Their disadvantage is the higher cost, due to the increased complexity of their circuits. One could argue also that the scaling behavior of a cross-bar system may prove expensive, as it scales as $O(n^2)$ and requires $2n-1$ new switching elements for every new processor added to the system. However, this is rather a technological limitation. Cross-bar systems require also different techniques to maintain the caches consistent to memory, than those used in bus based systems.

4 Performance Aspects and Evaluation Methods

Symmetric multiprocessor systems (SMP), offer a high capacity computing environment that applications can benefit from. In some particular cases additional performance with respect to uniprocessor systems (UP) can be achieved. In an SMP system, significant support from the side of the operating system is also required to ensure its own correct operation and offer to the applications a minimum set of the necessary primitives to utilize the parallel resources.

In this chapter we analyze the performance issues of SMP systems and their operating systems, in order to identify their relevance in the CMS DAQ architecture. The performance evaluation methods applied to computer systems and in particular for the CMS event filter farm, are discussed at the end.

4.1 Performance of SMP Systems

The performance of applications executed in MP and particularly in SMP systems, is not only determined by the performance of each building module of the system (*e.g.,* processors), but also from numerous other factors that emerge from the concurrent execution environment.

SMP systems offer larger computing capacity than the UP systems. At any time, different processors may execute unrelated applications. This kind of parallelism is very often referred to as *independent parallelism*. This is the minimum gain one can have from an SMP system. Very often it is met in large scale interactive systems. For independent or job level parallelism, usually little performance is required from the interconnection network apart from the typical needs of memory accesses.

Finer grained parallelism can be achieved by letting processes to have more than one *threads of control*, executing concurrently on different processors. This kind of applications are called *multi–threaded* (MT) and the type of parallelism is referred to as *medium–grained parallelism*. A thread of control is a programmer’s abstraction, used to represent a program’s code entity that can be scheduled for execution separately. Usually the first MT program running on an SMP system is the operating system (OS) kernel itself. MT applications, often require the exchange of synchronization messages between their threads. In MT applications, the explicit specification of the different threads is done by the programmer, thus making more complex their development.

The various applications running in CMS event filter unit (EFU) are expected to benefit from both independent and multiple thread parallelism. The independent parallelism is trivially satisfied by the number of applications. On the other hand, both event filtering algorithms and the software specific to interface with the external world of the EFU (switch or computing services) can be easily designed to make use of multiple threads. The MT programming paradigm, is the choice for the design of the switch-to-farm (SFI) emulator described in Chapter 6.

The additional to memory access traffic that is emerging from the thread synchronization, indicates that a high performing system interconnection is needed. The interconnect must accommodate also the synchronization traffic generated by the processors in order to keep their caches consistent.

4.1.1 Interconnection Performance

The choice of the interconnection topology and technology constrains the performance characteristics of an SMP systems. For instance, a bus based system can electrically accommodate only a limited number of processors and memories. As the number of processors increases, apart from the possible electrical limitations, contention due to the many bus transactions will occur causing the processors to stall. Consequently, this will reduce the processor utilization and the performance of the applications running on the system.

The amount of cache synchronization traffic may take away a substantial fraction of the available bandwidth of the interconnection. Consequently, this will limit the scalability of the system. Typical bus based SMP systems have six to eight processors, while few of them by utilizing multiple buses can reach up to 30 processors. Non-broadcast type of interconnections like crossbars can be much less exposed to contention. Overheads will still exist and a wide-span of parallel applications may run on SMP systems with crossbars, however the many independent connections offered by such topology minimize contention.

As Patterson and Hennessy note [Henn90], Input/Output (I/O) has been the orphan of computer science. Modern systems can accommodate a variety of I/O devices, from a keyboard, or a mouse, to high performance disk arrays and network interfaces. Perhaps, this is the main difficulty having a general solution for interfacing I/O devices to a computer system. I/O devices are usually connected to a system through an I/O bus and I/O to system bus controller. There are also cases like the MBus [Kell91], where I/O devices are connected directly to the system bus.

There are two ways of doing I/O transfers: programmed I/O (PIO) and direct memory access (DMA). With PIO, the processor is directly involved in moving data between the memory and the I/O device. In other words, the processor is accessing the device the same way it does with any memory location. Compared to the processor cycle, PIO operations are expensive and will stall the processor until the transfer has finished. With DMA, additional devices—the DMA controllers—are in charge of making the transfers between memory and the I/O device. This is achieved by stealing bus cycles from the processor. DMA transfers are initiated and finished by the processor without any other processor involvement during the data transfer. If the processor is making at the same time with the DMA transfer many memory accesses, it can stall until it is granted to use the memory bus. DMA transfer speeds are much higher than those achieved with PIO and more effective because of the minimal processor involvement.

The I/O performance of a system can be characterized by the throughput (or bandwidth) and the latency. I/O throughput refers to how much data can be moved in a unit of time, while latency refers to the time interval between the initiation of an I/O request and its completion.

A high performance SMP system, is very likely to have high I/O demands due to the number of applications running concurrently. Factors limiting I/O performance may lie anywhere in the path between the I/O device and the memory modules. The slowest device in that path, will determine the upper limit of I/O performance.

Most common throughput limitations, emerge from the I/O device itself and the I/O bus. Buffering the data stream to the I/O device can increase bandwidth, however it will degrade the latency of the device. Split-transaction bus designs can be used, to increase the bandwidth of an I/O bus. Buses supporting split transactions (also called packet based), can accommodate more than one I/O operation and thus not blocking during slow operations. However, such buses have higher latency time, because of the multiple packet nature of every request.

Some of the systems in order to have consistent the I/O transfers, implement a separate MMU to be shared by the I/O device controllers. That way, DMA transfers are done from and to virtual addresses. In addition to the MMU, they may have separate cache dedicated to the I/O devices. In such cases, the I/O coherency is solved with the usual methods of cache coherency, as they will be discussed in the following paragraph.

The I/O performance is of crucial importance for the EFU. The sustained bandwidth of the SFI must be sufficiently high to accommodate the needs of a particular farm configuration. I/O Latency is also relevant to the SFI performance, because of the high rate of messages send to the event manager (EVM).

SMP systems with processors that have private cache memories connected to them, need a mechanism to maintain their caches consistent during the application execution. This is usually accomplished by a variety of techniques implemented in hardware that stay transparent to the applications.

Data that are only read by the system’s processors do not need any special handling. The cache consistency mechanism will be needed once a processor decides to modify the shared contents of its cache. In that case, other processors’ cache contents will have either to be updated or invalidated. Also, if a processor references stale memory data (cache miss), the cache with the valid data has to be able to supply it. The unit of data that caches exchange between each other is a cache line and this is done using cache consistency protocols over the system bus, or interconnect in general.

We will distinguish two major categories of cache consistency protocols, the snooping and directory based protocols. *Snooping protocols* are based on the ability of each cache to snoop the transactions of other cache memories and thus they imply the existence of a broadcast type interconnect *e.g.,* a bus. Many snooping protocols have been proposed, but in general all fall into two main categories: the write-invalidate and write-update. The *write-invalidate* protocols imply that a cache will broadcast to the other caches an invalidation once it modifies shared data. Caches implementing *write-update* protocols will broadcast to the other caches the new value of the shared data.

Common write-invalidate protocols are the write-through invalidate, write-once and the MESI protocols. The first two are variations of the write-through and write-back cache update policies. The MESI protocol derives its name from the first letters of its states Modified, Exclusive, Shared and Invalid. By adding the additional information of ownership of a cache line, MESI protocols can be more efficient than other protocols as there is no invalidate broadcast when a processor loads data in the exclusive state.

Directory-based cache consistency is used in SMP systems with crossbar interconnections, where snooping cannot take place as there is no broadcast. The information needed to maintain cache consistency is instead stored to the main memory modules into *directories*. Every directory contains information on which cache has which memory line and the state of the cache. A disadvantage of directory-based cache consistency protocols (also often attributed to crossbar based systems) is that additional memory is required to store the directories. Typical values of the directory size are around 10% of the main memory.

The hardware consistency protocols add significant complexity to an SMP system, frequently resulting in higher cost. Other solutions could be that the compilers or the OS take care of the cache consistency. This will result into SMP systems with much less latency of memory operations. However, it seems that the transparency gained with hardware maintained cache consistency is such a big advantage, that is enough to overcome the additional cost.

The cache consistency protocols, will add more traffic on a bus based system. An MT application, apart from the explicit synchronization that it may require between its threads, it will generate additional traffic on the interconnection, to keep consistent the data shared by the processors it runs. Some of this traffic can be just overhead due to false sharing of cache lines putting additional load to the interconnection. False sharing will occur when two processors store data into different memory locations, but happen to be in the same cache line.

Cache coherency might be also relevant to the I/O performance. On output requests, the I/O device may access stale data from memory and conversely during input requests the CPU may get stale data. In the case of a write-through secondary cache, the I/O transfers are transparent. With write-back caches, either the OS has to flush the cached data or the hardware.

Cache performance is very important for the EFU. At a first glance, the fact that the cache system is transparent to the user level application, leaves the impression that very little can be done in order to have a cache performance aware application. There are two areas of concern of the cache performance. The first is the path of data between the SFI and the main memory. The SFI is essentially an I/O device of the EFU that could be thought of doing DMA transfers to main memory locations and one memory copy (or in the worst case more than one). The second is the performance effect of the cache for the particular set of LV2 and LV3 algorithms. The data received by the SFI is expected to generate some synchronization traffic as it will access a closed set of buffers. However, this traffic is expected to be minimal (will not trigger expensive memory update cycles) as the event filtering algorithms are not expected to modify the LV2 or LV3 data received by the SFI. The result of LV2 and LV3 filters will be a set of new data structures, appended to the original data. On the other hand, the LV2 and LV3 filtering applications will also occupy cache locations. It is not known as of today any estimates of their sizes. Due to the nature of event processing, we can see that a processor cache that can hold both the event under processing and the active part of the event processing algorithm will be a good choice. From the total size of an event (1 MB) we can estimate that caches with size larger than 1 MB will be needed.

4.1.2 SMP Operating Systems Aspects

The design of an OS for an SMP system has to take advantage of the additional performance offered by SMP systems. Many of the OS services can be executed in parallel, while some of them can be conflicting and special care must be taken to ensure the correct operation of the system. In the trivial case of a UP OS running on an SMP system, not only it will not be able to exploit the parallel environment of an SMP, but also it will inhibit the performance of applications running independently. This is because a single copy of the OS is available to all applications.

An OS designed for SMP systems has to offer the necessary services to its applications, so that they can also benefit from the parallel environment. In that respect, if the OS itself is viewed as a special kind of a process, it has to implement techniques similar to those met in parallel processes, in order to take advantage of the parallel environment.

The existence of a carefully designed MT kernel is a very important requirement for a high performance SMP system. In such kernels their system calls are re-entrant and hence, many applications running on different processors can make calls to them without being blocked. System functions like interrupts are also re-entrant and many of them can also be served by the system's processors. The design of (or conversion of a single threaded to) an MT kernel needs new abstractions for synchronization and data locking, that require some minimum support from the underlying hardware. This additional functionality is required to resolve the points of conflict between concurrent calls of the same OS function. Independent of the internal concurrency that can be achieved by the OS, accesses to several data structures *e.g.*, the process table, cannot be concurrent and must be serialized.

Different threads during their execution need to access shared data. In an SMP environment, even with sequential memory model, the ordering of simultaneous accesses to the same memory location cannot be deterministic. The lack of determinism results in race conditions. The code segment that operate with data that race conditions may occur is called *critical section*. Avoidance of race conditions is usually done by instrumenting the critical section with mutual exclusion (or mutex) artifacts. Similar race conditions occur even in UP systems, when asynchronous events like interrupts may cause the modification of data structures used by the interrupted code. However, such cases can be easily avoided in UP systems, by rasing the processor interrupt level when critical data structures are modified. This technique is not applicable in SMP systems, where more than one processor can alter the critical section.

The underlying hardware mechanism that supports the mutual exclusions is usually a set of special processor instructions for atomic read-modify-write operations. Such instructions guarantee that a read and a modify step can be done atomically *i.e.,* without interruption from another processor. An example of such instruction is the *ldstub* instruction of the SPARC processor.

Other types of locking techniques are the so called spin locks (or busy wait), conditional locks and condition variables. A *spin lock* is acquired by a processor before entering a critical section. If a second processor tries to access the locked resource, it will spin in a loop (or busy wait), until the lock from the first processor is released. Using a *conditional lock* instead, a processor will attempt once to acquire the lock and if it fails it may continue execution, rather than spinning. The *conditional variables* are used to suspend execution until a particular condition becomes true. The execution of threads with critical sections that attempt to access the shared resource will be suspended if the resource is not available. Once the resource becomes available, a broadcast signal will be sent to all threads previously blocked, in order to resume execution. Conditional variables are a very powerful abstraction. They are used in conjunction with mutex locks.

The simplest type of synchronization is the Dijkstra *semaphore* [Dijk68]. It is represented by an integer number, on which two atomic operations are defined, and a queue holding information of blocked threads. The *P* operation will decrement by one the value of the semaphore and will block the execution of a thread if the resulting value is less than zero, otherwise the execution is continued. The *V* operation will increment by one the value of the semaphore and will resume the execution of a thread previously blocked, if the resulting value is less or equal to zero. Semaphores are the basis of many implementations of mutual exclusions and spin locks today.

The choice of the right primitive is a very important performance consideration. For instance, spin locks on large critical sections or when slow operations like I/O take place, may degrade systems performance, because useful processor cycles are wasted. As the amount of synchronization and locking primitives increases in a program, additional overheads will occur, degrading its performance. Conversely, at the design stage of a parallel program, the choice of shared data structures and their implementation, may play an important role. A mutual exclusion on a heavily used shared data structure, will result in contention (or a hot spot) inhibiting performance.

Another effect that may appear in a concurrent environment is that thread A holds locks of resources required by thread B, while B holds locks required by A. If one thread attempts to acquire a lock on resource locked by the other thread, a deadlock situation will occur. Deadlock situations are never resolved and if they occur in the OS kernel any activity of the system will cease. If there is activity on the system, then we have a livelock. Livelock will occur if an event expected to happen, never happens but other threads are running on the system.

Careful design of MT applications, to order correctly their resource locking, seems to be the only way for deadlock avoidance. There exist methods for deadlock detection and prevention, but not all possible cases can be handled by them. Deadlock prevention mechanisms are usually expensive in terms of resources and contribute to overheads.

In an SMP system, it is possible to schedule not only entire processes, but also individual threads. The OS maintains different priority queues, all of them fed to a common pool of system’s processors. Tasks scheduled for execution are placed into one of the queues, depending on their assigned priority. Different methods have been proposed for thread scheduling. Amongst them are self-scheduling, dedicated processor assignment and gang scheduling.

With *self-scheduling*, ready to run threads are placed in a global queue from which processors select one thread when they become idle. Obvious advantage of self scheduling is the even load distribution among processors. However, the global queue will pose serious contention problems, when more than one processor attempts to select a thread. Also, MT processes with many threads that communicate frequently, will not perform well as the probability of having many threads running is relatively low.

With *dedicated processor assignment*, each process is assigned to one processor for execution. Such scheduling policy has the advantage that switching between the different threads of a process will add very little switching overheads. The gained performance comes from the fact that very little or no processor time is wasted for thread switching. Also the processor’s caches will already contain data of the same process when its next thread will run and expensive cache flushes and fills will be avoided. However, in situations where processes block on I/O operations, processor starvation may occur because the scheduler's queue has tasks ready to run and processors are idling. A variation of this scheduling method is the *gang scheduling*, where all threads of a process are simultaneously scheduled to run.

The performance of different scheduling methods, heavily depend on the type and the number of the processes running on the system. What may be fair for one type of applications may not be for another.

Relevant to scheduling, are the real-time characteristics of an OS. Various definitions of real-time exist, depending on the performance needs of the applications. We will limit the definition of real time behavior of an OS, only to its ability to respond to external events in as much as possibly known time intervals. Better real time behavior will mean lower response time to an event (*e.g.*, an I/O interrupt) occurring to the system.

The scheduling policies used in a system will influence its real time behavior. In an MT environment (both the applications and the OS kernel), the flexibility of the scheduler to decide to which tasks to assign higher priorities, plays a major role. As an example, consider a thread A with high priority accessing an I/O device, which is busy for a thread B of lower priority. The execution of thread A will be suspended and even worse, it may be delayed unnecessarily if a third thread C with priority higher than that of B exists, as it will preempt B. This problem is known as priority inversion. A common solution to cope with priority inversion situations is *priority inheritance*, *i.e.*, in the previous example, the lowest priority thread B will be given the priority of thread A, so it can release as soon as possible the resource blocking thread A.

Real time behavior of a system, will be limited by the granularity of the system clock. The number of clock ticks in one second in the best case will set the lower limit of the clock granularity. The clock granularity will limit the precision of events happening into the system. Modern SMP systems have clock granularity of few hundreds of nanoseconds and some of them, like the Silicon Graphics systems even have 21ns.

The performance issues related to the concurrency of an OS that were described above, are crucial if an SMP system will be used for the EFU. High performance interfacing to the SFI will require the presence of efficient (low-overhead) synchronization primitives as we will discuss in Chapter 6.

4.2 Performance Modeling and Evaluation

During the design phase of computer systems, a wide variety of technological choices are available to the systems designers. The decisions of which options will be used in a computer system, can result in systems with very different characteristics. Amongst the most important characteristics are the performance and the cost of the final product. The performance evaluation of computer systems during their design phase is today a crucial part of their design. It can lead the evolution of a computer system through the shortest path towards the desired performance and cost.

Performance evaluation is equally applicable after the design of a computer system. The various alternative systems can be compared and decisions on the suitability of one or another system can be taken on scientific grounds.

The performance evaluation techniques that are a common practice today, can be classified in two major areas. These are modeling and measurements. The modeling techniques are mostly applicable to systems that are under design and hence not physically available. Measurements imply that the system under study is available and its performance under different types of workload can be evaluated.

4.2.1 Mathematical Modeling

A model of a computer system is an abstract representation of its functionality in terms of mathematical relations. The performance of a computer system is described through a model containing a set of parameters describing the system and their values for a particular study.

Depending on the desired level of abstraction, a mathematical model can be able to describe a wide variety of systems in a general way. Hence mathematical models are a very powerful tool for the performance evaluation, as the results from them can be applicable to a more general class of computer systems.

A model can be deterministic or probabilistic. A deterministic model can always reproduce exactly the same behavior of a system for the same set of input parameters. In a probabilistic model, the system behavior is described in terms of probabilistic macroscopic phenomena that take place inside the system. Therefore, a probabilistic model can give different results in successive solutions for the same set of parameters. Probabilistic or stochastic models are very advantageous, because of their ability to describe a system with much less details built into them. More details of the system may be impossible to know and hence difficult to construct a model for it.

An important consideration when choosing an analytical model is its complexity. The solution of a model for a desired set of its parameters, can be obtained analytically for simple models. However, for more complex models, analytical representation of the results might be impossible to obtain. In that case numerical solutions can only be obtained, making the model cumbersome and difficult to use.

A class of stochastic models that are very popular in performance evaluation are the *Markov models*. They are named after the russian mathematician A. A. Markov, who defined them in 1907. Their popularity is mainly due to their limited mathematical complexity. A model is called *Markovian* when the next state of a system it describes, depends only on its current state. Markovian models can describe stochastic processes that their states can be either discrete or continuous. Also, their dependence on the time of the system evolution, can be either discrete time or continuous. A discrete-state Markov processes is called Markov chain and the values of its states are either finite or countable. A sub-category of the Markov chains are the birth-death processes, where the next state of a system can be only one from its neighboring states. A discrete-time Markov chain can be described through a transition matrix that give the transition probabilities of between the different possible states.

Another class of mathematical models are based on queueing theory [Klei75] [Klei76]. A *queue* is a system that customers or jobs are arriving to receive service from one or more servers. When the queueing system servers are busy servicing incoming jobs, newly arrived jobs before they can receive service, they must wait until a server becomes free. A queueing model has as parameters those related to the job arrivals and servicing. The job arrivals are described by the *interarrival time* τ and the *mean arrival rate* λ. Similarly the job service is described by the per job *service time* s and the mean *service rate* per server μ. In a probabilistic queue model the above variables are randomly distributed variables. Queueing models are classified according to the combinations of their interarrival and service time distributions, and their service characteristics *i.e.,* the policy of selecting the next job to be serviced.

Queueing models have received an extensive attention over the past years. They elegantly can provide analytic expressions of the various performance variables for a wide variety of queue types.

Many independent queues can be connected together to form a *queueing network*. In a queueing network, jobs that finished service from one queue are forwarded to the next, queue according to the network connection. A queueing network can be open or closed, depending on whether the jobs that finish service from the last queue of the network are entering the network again or they leave the network.

Queueing networks are very well suited to describe computer systems. Based on them, very simple models can be constructed and easily evaluated. Their main advantage are related to the easiness that computer system models can be constructed out of the individual functional units of such systems.

Simplicity advantages equal to the queueing networks, are offered by the *Petri nets*. They were initially introduced by C. A. Petri in 1962. The inherent generality of Petri nets makes them applicable in a wide range of modeling problems, spanning from performance evaluation and communication protocols up to human factors modeling [Mura89].

A model based on a Petri net describes a system in terms of its possible states and the transitions between them. A Petri net is a graph consisting of two kinds of nodes; the places and the transitions. Arcs are used to connect places to transitions and vice versa. A particular state of the net is called marking. A marking assigns to each of the net’s places a number of tokens. Tokens are used to represent that the condition assigned to a place is true. A transition can fire only if the places connected through arcs have tokens. This is a necessary but not sufficient condition for a transition to fire. Once a transition fires, the number of needed tokens to fire are removed from the input places and assigned to its output places.

Petri nets are very well suited for the study of problems associated to concurrent systems [Pete81]. Stochastic extensions of Petri nets have been used for studying the properties and problems of multi-processor systems [Mars88].

Among the most important studies of Petri nets are those related to their reachability, boundedness and liveness. Reachability refers to the possibility to identify if a certain marking of net belongs to its set of possible markings. Boundedness refers to the property of net that the tokens found in each of its places do not exceed a certain number for any reachable marking. A Petri net is said to be live, when any of its transitions can fire independent of its current marking. More relaxed conditions of liveness are also common.

4.2.2 Simulations

Besides the mathematical models, the simulation models are also particularly popular in performance evaluation studies of computer systems. A simulation model is a computer program that describes the behavior of a system and the workload under study, with the aid of algorithms [Fish95]. The various performance indices of the model are obtained by sampling the simulation program execution.

Similarly to the mathematical models, simulation models are classified as deterministic and probabilistic. The systems they describe, can have states that take discrete or continuous values. Also, the time dependence of the model can take discrete or continuous values. Computer system models are usually discrete-state, continuous or discrete time. They are built using either high-level programming languages like C++, C and FORTRAN or specialized simulation languages as SIMULA and SIMSCRIPT.

The discrete-event simulation models are usually built by small modules, each representing the functions of a sub-system in the model. Each module interact with the other modules by means of simulation events. These events are scheduled to occur in various points of the simulation time and their occurrence triggers an action in the respective module. Discrete-event simulations, require the existence (or the implementation) of an event scheduler and a simulation clock.

The event scheduler manages a linked list of the various events waiting to happen. It receives events from the individual modules of the simulation and schedules them to occur at a given simulation time. The simulation clock is usually a global variable of the simulation program representing the elapsed simulation time and it is advanced by the event scheduler. There are two approaches to advance the simulation clock. With the unit-time approach, the simulation clock is advanced by one predefined unit and then the event scheduler checks for any events that might occur. With the event-driven approach, the scheduler advances the simulation clock to the time of the next earliest occurring event. The event-driven approach is the mostly used in simulations of computer systems.

Every module of the simulation that may accept simulation events, requires an event-handler routine. The event-handler performs various actions depending on the type of the incurring event and the current state of the module. In response to the events, the event handler may generate other events to one or more simulation modules.

The event scheduler of a discrete-event simulation is the most often executed part of the simulation. Its implementation must ensure that incoming events are properly ordered in time. A large number of simulation events may require a significant amount of computations to be performed when a new event arrives or the next event to occur is searched. The efficiency of the different data structure types that hold the simulation events, depends on the number of events held in the data structure.

Throughout the simulation, the performance indices specified by the simulation model are sampled. Their values are reported at the end of the simulation. The sampling nature of the those results, requires that proper confidence intervals are computed for each of the performance estimates. The width of the desired confidence interval of a performance index, is also relevant to the simulation length. Methods like the batch means can be used for the estimate of variance and also the removal of the transient part of a performance estimate.

Probabilistic simulation models require the generation of random variates according to a specified random distribution, for the different model variables. Random variates are generated with the aid of random number generators. The choice of a random number generator is crucial for the correctness of the simulation results. A random number generator, must provide independent and uniformly distributed random numbers. The period of the random number generator must be chosen to be long enough, so that the generated numbers during a simulation are unique.

Once a simulation model has been built, the results obtained by it must be validated and verified. Model validation refers to whether the assumptions used to build the simulation model are reasonable representatives of the system under study. Model verification is the procedure to prove that those assumptions are correctly implemented in the model. Both validation and verification are probably the hardest part of simulations projects. Several techniques for validation and verification exist [Jain91]. They all require thorough understanding of the simulated system, the simulation program and the data analysis methods.

The popularity of the simulations in performance evaluation is mostly due to the offered ability to describe systems in various levels of detail. Also, they can advance independent from the availability of the real system and provided that they can become available early enough, various design alternatives can be evaluated and trade-offs resolved. However, they are often attributed the disadvantages of requiring a long development time, difficult validation and verification and sometimes a lot of processing power to run them.

The method that we have chosen for the performance evaluation of the CMS even filter farm (EFF) is using system models and discrete event simulations. The main arguments justifying our choice are: (a) the flexibility that simulations offer to model a system depending on the desired level of detail, (b) and the mode of operation of a device, and (c) the ability to describe the system scaling for various EFF implementation scenarios. As an illustration of (a), we can consider the case of the RDPM. The RDPM is a custom designed device, that its operation must be reproduced as close to the real system as possible. Precise simulations of the RDPM are very useful for the RDPM development. A simpler model of the RDPM can be also useful to understand the behavior of the rest of the EFF system. Similar arguments to (b) are valid for the event manager (EVM). At the early stage of design, only the main concepts of the EVM operation are known. Particular implementations of its operation can be tried out using simulated models of the EVM. Once the design is finalized, different EFF implementation options can be evaluated with regard to their scaling ability (c).

The analytical formalism is not excluded but it is expected to complement the evaluation studies where it is possible. Simulation verification is an example of a case where analytically obtained results can be compared with the results obtained by the simulation.

A simulation tool built on the grounds of reproducing the system behavior in parallel to the design phase, has the additional advantage that it will be capable to describe the real system after its construction. This feature can be very useful in situations where the effect of system changes must be first evaluated before the system is modified.

4.2.3 Benchmarks and Measurements

Measurements performed on real systems, provide an accurate means of comparing performance under various workload and system operating conditions. Various types of workload can be used for measurements, exposing the performance of the system under study in respect to one or more performance indices. For the comparison of computer systems with different characteristics, the same workload must be used. The workloads chosen for this task are called benchmarks.

There are several benchmarking suites available today. Some of them measure the overall system throughput relatively to a previously system, for a mixture of various kinds of typical workloads met in computer applications. Others, often called micro-benchmarks, are well suited to expose the performance of the individual sub-systems like the memory hierarchy, the I/O units etc., or even individual services of the operating systems as are the system calls and context switching. An example of a micro-benchmark is *lmbench* [McVo96]. It focuses on latency and bandwidth measurements, that the various building blocks of a wide range of applications are using. More specialized benchmarks focus only on one subsystem, as is the STREAM benchmark [McCa95]. STREAM measures the memory bandwidth for four different patterns of memory references.

Memory bandwidth measurements are very useful for the SFI emulator evaluation that will be performed in Chapter 6. Results obtained by the STREAM benchmark will help us to determine the available memory bandwidth of the system under study.

The choice of an appropriate benchmark is an important consideration when the performance of computer systems is compared. The results of comparisons using an inappropriate benchmark may be misleading for the actual performance of the systems in a real operation environment. This has resulted to less common benchmarks but much more representative of the anticipated workload. An example of such specialized benchmark suite is the Cern Unit benchmark. It is a collection of typical high energy physics processing and analysis applications, which involve a mixture of heavy floating-point and integer computations and memory references, but with very little I/O.

The results of measurements with benchmarks are very sensitive to many parameters of the system under study and the benchmark itself. Meaningful results require that the measured system is well understood *e.g.,* memory cache effects, compiler optimizations, configuration of the measured system etc. Also the internals of the benchmark need to be well known, as for instance the effects of sampling and monitoring that might be used to obtain the performance figures, may affect the final results.

4.3 Summary

We discussed the performance issues that are typical to MP systems. Amongst them, we distinguished the architectural related issues and the impact of the OS. In the case of an EFU based on an MP system, the factors affecting the performance of the I/O sub-system and the memory sub-system are crucial. The architecture of the interconnect is also important in order to provide the necessary performance scalability. The impact of the OS can be characterized as the means it provides for the exploitation of the SMP architecture (threads, synchronization primitives, etc.) and its own design taking advantage of the parallel architecture.

We discussed also the performance evaluation methods commonly applied to computer systems. For the purposes of the performance evaluation of the EFF, we believe it is appropriate to use a discrete-event simulation tool. Our choice is justified because of the needs of system and model design flexibility during the design phase of CMS, and the possibility to have a working model of the EFF during its operation.

5 Farm Simulations

A full design of the CMS DAQ system, requires that several farm configurations and setup conditions are studied and evaluated. Also, until the construction of the DAQ system itself, its design has to be flexible enough to handle any changes of the requirements, as well as the appearance of new technologies that will influence it. A possible approach to achieve this is through the study of prototype systems. We propose system simulations to be used as a complement to prototyping. System simulations can give invaluable insight on the studied systems in cases when prototyping is either not possible or not economical. Another important issue is that under certain circumstances, simulations can also describe the overall system scaling behavior.

In this chapter, a simulation tool of the event filter farm (EFF) is introduced. It implements a relatively general model of the functionality of the major DAQ components and has the ability to describe their behavior at different levels of detail.

5.1 DAQ Simulation Model Description

The modular design of the CMS DAQ system, easily permits a functional decomposition into, either separate subsystems or replicated subsystems. For instance, the RDPM used for LV2 and LV3 event fragments are essentially providing the same functionality. They can be modeled once, and instances of that model can be used into the overall simulation model. That observation gives great flexibility in the design of a farm simulation model, as the different components can be implemented at different levels of detail and at different times. Figure 5.1 illustrates this module decomposition. Based on that observation, we build the model of the DAQ system using a set fundamental functional units *i.e.*, the EVM, the RDPM, the switch, the SFI and the EFU, that interact with each other by exchanging action messages. Those messages shown in Figure 5.2, trigger an appropriate action (eventually another message) in the device they are destined to.

The LV1 trigger, it is described as a generator of DAQ events. It sends LV1 trigger messages to the LV2 and LV3 RDPM and to the EVM, with a distribution of the interarrival time τ_{LV1} defined at the beginning of the simulation. The LV1 trigger messages signal to the RDPM a front-end read and in the EVM the presence of a new DAQ task. They are also tagged with a serial number that identifies them uniquely during a simulation run.

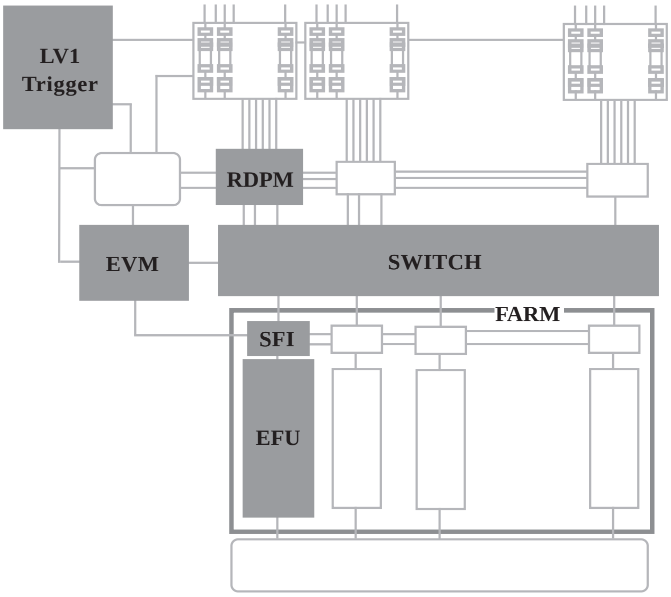

Figure 5.1: Functional decomposition of the DAQ system

The RDPM are modeled (according to the diagram of Figure 2.2) as a set of FIFO queues for the IEq, IENq, FRq, FRB and OED. They contain the necessary event generators and event handlers for the SeqIN, TS and SeqOUT. The PBT and the Memory are linked lists of the input events. The readout system (*i.e.,* the RU and FED) is not modeled, as it is considered to be outside of the scope of this simulator. We assume that the input from the FED to the RDPM occurs with little or negligible dead time. This dead time is accounted for in the readout time, simulated by the RDPM.

Because of the need of a detailed description of the RDPM devices, an implementation of a simulator using 1,000 instances of RDPM described in detail, could be very complex and very resource-demanding. Therefore, we have decided to implement the simulator using only two instances of the RDPM. Each instance represents an RDPM group serving either the LV2 or the LV3 event parts. In order to be consistent with the DAQ system operation, a LV1 trigger arriving into an RDPM will start the readout of an event fragment and not of an event part (LV2 or LV3). However, outgoing messages from the RDPM to the switch will represent the sending of a LV2 or LV3 event part, depending on the sending RDPM instance. We believe that this simplification is reasonable because without compromising the simulated functionality of the DAQ system, the results obtained for the RDPM itself are still valid.

In the same context of the RDPM simplification, we have chosen also not to fully simulate the event builder switch. The switch is modeled as a simple delay server with parameters that have been obtained from a separate detailed simulation [RD3195]. Incoming messages from the LV2 or LV3 RDPM will be forwarded to the requested destination in EFF with a delay proportional to the event part size, as given by the detailed simulation results. With the above two simplifications, the complexity of the simulator is significantly limited. Obviously the price to pay for those simplifications is that we will not be able to study the switch behavior in the simulated systems.

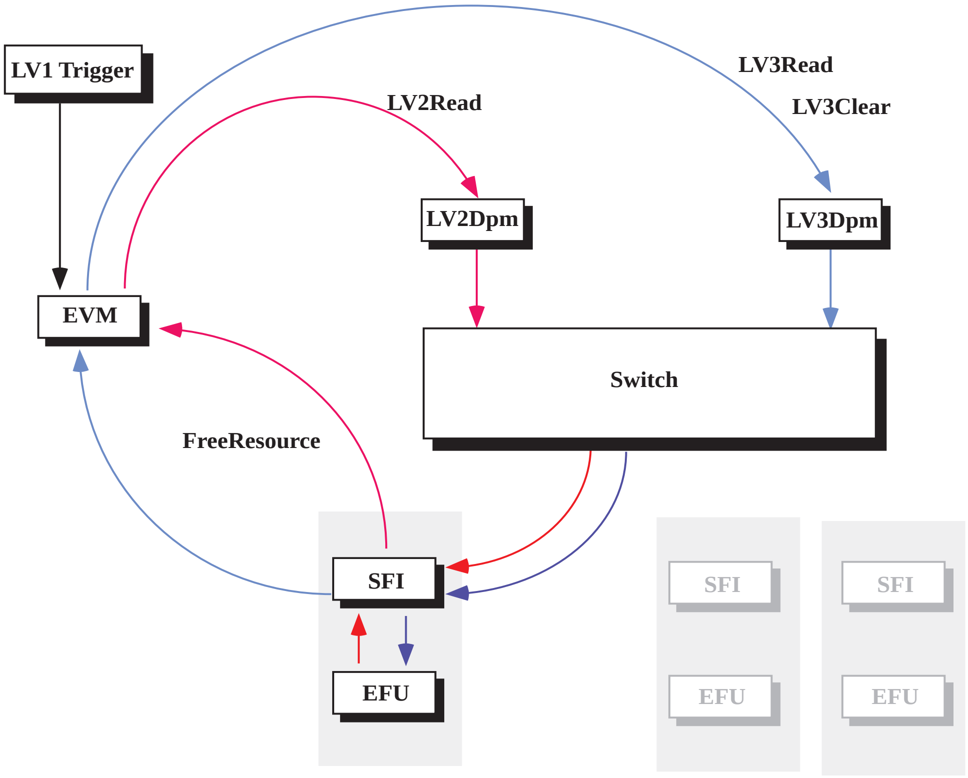

Figure 5.2: Overview of the DAQ simulation model

The EVM is modeled as an event handler of LV1 trigger messages and messages arriving from the EFU requesting to process a new LV2 event or a LV3 event. It contains two queues, one holds the requests for LV2 event parts and the other requests for LV3 parts. Its operation consists of assigning LV1 triggers to LV2 requests from the EFU, and forwarding requests for LV3 event parts to the RU. Also in case of a rejected LV2 event, it sends a clear message to the LV3 RDPM (Figure 5.2).

5.1.1 Farm Model

In the studied simulation model, we have introduced an abstraction of the overall EFF which is used for the coordination of the individual EFU. It is convenient for routing messages received from the switch module to the EFU, and collect global statistics. On every new LV2 or a LV3 event, the farm will examine it and forward it to the SFI which has requested it.

Each farm node models an SFI connected to an EFU. During the initialization phase of the simulator, the farm object will construct as many SFI and EFU pairs as specified by the studied configuration.

The SFI are modeled as a collection of buffers (Figure 5.3) each representing an available filter resource (FRS). In the EFU model, a DMA engine is foreseen to simulate the transfer of the incoming event part from an SFI buffer to a local to the EFU memory buffer. LV2 and LV3 event parts arriving from the SFI, are placed into separate queues (LV2q and LV3q respectively) into the EFU memory. An additional queue (not shown in Figure 5.3) is used to hold pointers to available processors.

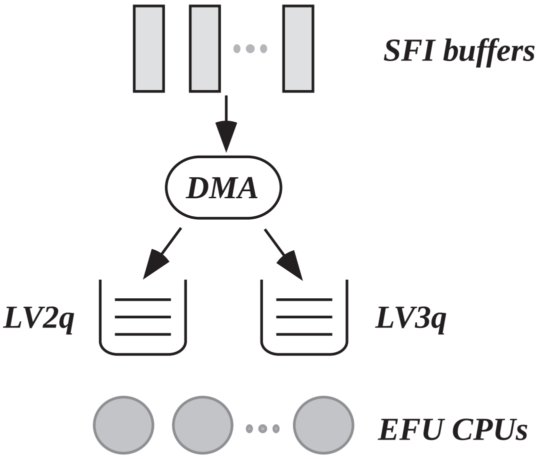

Figure 5.3: SFI and EFU model

The main idea of having buffers in the SFI and queues in the EFU, is to be able to unload the RDPM from pending LV2 requests and have the flexibility in the EFU to schedule its resources depending on the number of the outstanding LV2 and LV3 jobs. As it is shown in Figure 5.2, once an SFI buffer has been emptied by the DMA engine, it is considered free and a FRS request will be sent immediately to the EVM to get the next job. This has the effect that RDPM will always be stable (*i.e.*, memory will not overflow) given that enough FRS are available. Having the two queues in the EFU, instabilities can be detected by monitoring their length and corrective actions (*i.e.*, additional constrains on the filtering algorithm execution time) can be applied. The number of the FRS each SFI will initially communicate to the EVM, reflects the capacity of the EFU to handle incoming events. Less buffers per SFI are expected to result into increased memory size used in the RDPM and vice-versa.

This EFU model is designed to give higher priority to the LV2-trigger jobs than the LV3-trigger jobs. It has also the option to preempt LV3 jobs in favor of LV2 jobs when there are no free processors. With LV3 preemption enabled, the execution of the suspended LV3 job will be resumed immediately after the LV2 job that caused the LV3 preemption has finished. LV3 preemption is an interesting way to minimize the time a LV2 decision is taken. Less time to take a LV2 decision is expected to increase the system's throughput.

The effect of LV3 preemption can be characterized to be proportional to the ratio of s_3 (the LV3 processing time) over s_2 (the LV2 processing time). When this ratio is close to one, queued LV2 jobs will wait time proportional to their service time. In that case the system will appear like more LV2 jobs are present in it. In that case, eventual LV3 preemption might not be a good choice, if a possible preemption overhead is taken into account. However, when the ratio of the service times is around 100 (see Table 1.5) as in the case of the CMS DAQ system, queued LV2 jobs may have to wait before they are serviced, a time up to the LV3 service time. Additional waiting time will increase the LV2 decision latency. If LV3 preemption is enabled in that case, the LV2 latency will be minimized, but the LV3 latency will be increased by the number of times a LV3 job is interrupted on average, multiplied by the LV2 service time s_2.

Another factor to understand the impact of LV3 preemption is the LV2 rejection factor R_2. When R_2 is low (*e.g.*, in the order of ten) the number of LV3 jobs is high and vice-versa. This observation is very important as it points to the system stability condition when LV3 jobs are preempted. This can be expressed by the following relation.

$$\frac{N}{r_2} > n_i \cdot s_2 \qquad \textbf{(F 5.1)}$$

Where N —the number of EFU, r_2 —the resulting LV2 trigger rate (*i.e.*, the rate of LV3 jobs creation), n_i —the number of times a LV3 job is interrupted on average, s_2 —the LV2 jobs service time. The left term of the Formula 5.1 essentially represents the arrival time of LV3 jobs in each of the EFU *i.e.*, $1/\lambda$ while the second term represents the average service time of LV3 jobs *i.e.*, $1/\mu$. In other words, Formula 5.1 represents the stability condition $\rho < 1$ for the traffic intensity of LV3 jobs.

With LV3 preemption, the LV2q is expected to have only a few entries, resulting to a minimum LV2 decision latency (*i.e.*, the time from the LV1 event occurred, until the time the LV2 filter reached a decision). On the other hand, the LV3 decision latency is expected to be much higher than the real time needed by the LV3 filter to execute. Given sufficient memory resources and that suspended LV3 jobs do not accumulate, the increased LV3 decision latency is not considered as a problem.

5.2 Input Parameters

Some 20 different input parameters are used to describe the simulation model of the DAQ system. They can be divided into two categories: those that are directly related to the physics requirements and those that are used to describe the specifications and configuration of the utilized components in the simulated system.

Parameter	**Value**
LV2-RDPM block size	*1024 Bytes*
LV3-RDPM block size	*1024 Bytes*
LV2-RDPM memory size	*128 MB*
LV3-RDPM memory size	*128 MB*
SFI DMA speed	*100 MB/s*
LV3-part relative size	*0.75*

Table 5.1: Fixed simulation parameters

Into the first category fall the LV1-trigger interarrival time, the event size distribution, the execution time of the LV2 and LV3 filtering algorithms and their respective acceptances.

Into the second category, fall the ratio $S_{2/3}$ which is defined to be size the LV2 sub-event over the size of its corresponding LV3 part, the RDPM parameters and the characteristics of the switch, the SFI and the EFU.

For the simulations described in this chapter, some of their input parameters are fixed to the today's assumed values. These fixed input parameters are shown in Table 5.1.

5.2.1 LV1-Trigger Characteristics

The LV1-trigger can be described by a distribution function which models the expected physics interaction probabilities and the related to the readout and triggering hardware implementation. Precise information of the LV1-trigger distribution function is not known yet. However, it is reasonable to assume that LV1 triggers arrive in exponentially distributed time intervals, mostly because of the Poisson nature of the physics interactions. Another possible assumption could be a geometric distribution, which might be more realistic from the point of view of how the LV1-trigger and the global trigger hardware will operate. As a working assumption during the current design phase are consider two cases of the τ_{LV1}. One were it is fixed at 10µs as it is the design maximum of the system, and a second with a negative exponential distribution with mean at 30µs (a value slightly higher than that pointed by the current physics results, as shown in Table 1.4).

5.2.2 Event Size Distribution

A full event accepted by the LV1 trigger is expected to have an average size of 1 MB. The various sub-detectors contributions to that figure are shown in Table 1.4. The characteristics of the distribution of the event size can be obtained from detailed physics simulations of the different sub-detectors. The mean value and variance of the event size are parameters that are related to the luminosity $\mathcal{L}$ (*i.e.,* the amount of interactions per unit of time) of the LHC accelerator besides the number of channels of each subdetector. The value of $\mathcal{L}$ during the operation of the experiment will decrease as protons are colliding and increase again to its nominal value, when new protons are injected into the collider. That change of $\mathcal{L}$ over time, will affect the event size as the amount of particle interactions and hence the number of fired detector channels varies.

The farm simulation model itself takes very little into account the event size. It is only in the RDPM model where in combination with the average event fragment size and the block size of the RDPM memory, it has a significant effect on the total amount of memory that is in use. In any other sub-system of the DAQ the event size might contribute to the transfer latency (DMA transfer etc.). This contribution is very small compared to the latency of the switch. However for the detailed switch simulations it is crucial to have more precise figures of the event size.

5.2.3 High-Level Triggers Workload

The workload characteristics of the filtering algorithms are even harder to specify, mostly because they are not yet developed. It is known that the LV2 filter algorithm must have a bounded execution time so that the LV3 RDPM memory or the LV3q queues in the EFU have a reasonable size (below 100 MB). One possibility is to characterize the high-level triggers with a distribution function of the execution time (s_2 or s_3) needed to reach a decision (accept or reject). This seems convenient as it inherently describes the performance of the EFU. On the other hand, shapes of that distribution cannot be easily guessed as they are strongly dependent on the actual implementation of the filtering algorithms. One possible model could be that as the execution time elapses the probability to reach a decision increases. However, successive values of s_2 may be related to each other if the algorithms get biased from the LV2q or LV3q queue sizes in the EFU.

We will consider s_2 and s_3 with fixed values to 9ms and 900ms respectively. The choice of those values is based on rough assumptions on what they might be at the beginning of the experiment, for today’s preliminary LV2 and LV3 filtering algorithms.

Another important characteristic of the filter algorithms is the rejection (or acceptance) efficiency they achieve on they incoming data. We will characterize that effect with the rejection factor of a filter algorithm — R_2 or R_3 defined as the ratio of the input event rate over the output rate. It is obvious that R_2 is a crucial parameter of the LV2 filter algorithm as it directly determines the input rate of the LV3 jobs in an EFU. R_3 determines the output rate of the farm to the computing services and storage media. In the real system it is possible that R_2 or R_3 will not have a fixed value; instead they might vary according to the load condition of the farm. For instance, if the EFU output capacity permits, the LV3 trigger algorithm can be instructed to be more relaxed in its selection, so that more processing resources become available to LV2 triggers.

A general model of the workload characterization of the high-level trigger algorithms should also take into account the fact that during the operation of the experiment, newer versions of the algorithms may be developed resulting into either better algorithms and therefore more efficient (lower *s*), or into more detailed and slower ones (higher *s*).

5.2.4 Parameters of the RDPM

The set of parameters (Table 5.2) describing each of the LV2 or LV3 RDPM, includes the internal memory size and its block size, and the speed at which the RDPM is sending data to the switch. The internal queues of the RDPM are without a specified size (*i.e.*, infinite number of entries) and their occupancies are monitored during the simulation. The parameters used in that model of the RDPM are not directly related to the rest of the simulated system. Eventually, a simpler representation of the RDPM functionality *e.g.*, as a single queue could require a smaller parameter set to be defined. Using a detailed functionality model for the RDPM despite the simplified model of the switch, has the advantage of evaluating the required size of the various internal queues of the RDPM. Those results specific to the RDPM, after some further refinement, would be used for the construction of the RDPM hardware.

Parameter	**Typical Value**
LV2-RDPM Memory block size	*1024 Bytes*
LV3-RDPM Memory block size	*1024 Bytes*
LV2-RDPM memory size	*128 MB*
LV3-RDPM memory size	*128 MB*
DMA speed	*100 MB/s*

Table 5.2: RDPM parameters

5.2.5 Event Builder Latency

This simulation model assumes a switch for the event builder that it is modeled as a delay center, as it was discussed in 5.1. This simplification assumes that no internal or output congestion occurs and the switch has reached already a steady state at the maximum LV1 trigger rate of 100kHz.

The switch we take into account is a 1,000 × 1,000 ATM switch fabric that it built from 8 × 8 switching elements of 622 Mb/s (OC-12) speed. Such a switch has been simulated in detail for an environment similar to the CMS DAQ system and it is described in [RD3195] and [Teth95].

The latency distributions for the LV2 and LV3 event parts we will use, are shown in Figure 5.4 and Figure 5.5 respectively. It is shown, that the average switch latency to transfer a LV2-part equal to 250KB (17 ms) is almost 4,600 times higher than the physical media transmission time (3.6 ms).

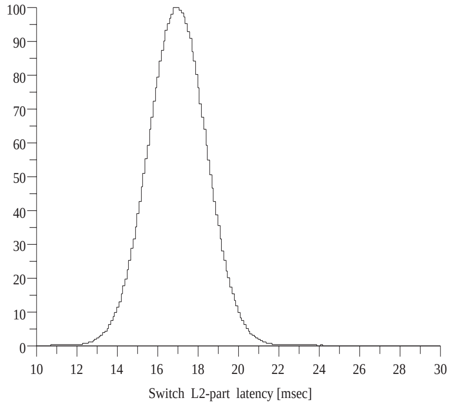

Figure 5.4: Simulated switch LV2 latency

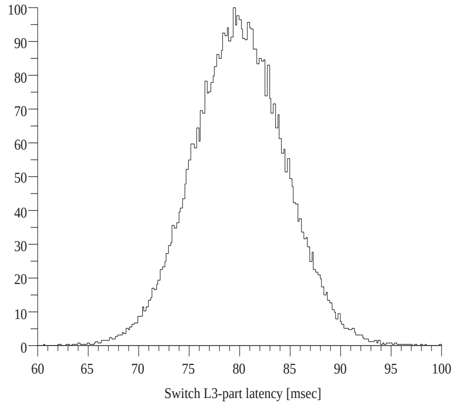

Figure 5.5: Simulated switch LV3 latency

The simulator may accept as input either a particular distribution for the LV2 and LV3 event parts latencies or it can interpolate from a given set of data-points that have been obtained from a separate simulation. The advantage of this approach is that as other switching technologies are considered, results from their simulations can easily be included into this simulation.

5.2.6 EFU Parameters

The EFU model requires only one configuration parameter, the number of processors. The EFU performance is not directly defined, but only through the average service time of the LV2 and LV3 filter algorithms. Another parameter is used to specify whether the for interrupting the LV3 trigger by the LV2 trigger.

5.3 Simulation Program

Originally the simulation tool was developed for the design of the RDPM with a relatively simple implementation of the rest of the system. Later it was expanded to a more detailed description of the rest of the system.

From the very beginning, a discrete-event, discrete-time object oriented model was used in the simulation tool, using the C++ language. The object design naturally fits into the functional decomposition of the farm shown in Figure 5.1. The individual functional units (shown in gray boxes) are once implemented in classes and then instantiated as many times as they appear in the system to be simulated.

The simulator engine, implements all the needed event handling mechanisms (*i.e.*, simulation clock, event lists etc.) into a separate library. The integration of the simulator engine and the classes describing the DAQ system is done by simply inheriting from the simulation library classes. Every class that might receive simulation events, has to inherit from an abstract message-handler class of the simulator engine. Additional function methods, are implementing the different functions required by the simulator.

5.3.1 CNCL Class Library

The core of the DAQ simulation tool is based on the Communication Networks Class Library (CNCL) [Steppl] [CNCL], developed by the COMNETS group at RWTH Aachen. CNCL was designed for the performance evaluation of communication protocols through discrete stochastic simulations. The design of CNCL itself is based on the NIHCL (National Institute of Health Class Library), GNU libg++ and the SIC (Simulation on C++) class libraries. Besides its OO features as tree-like class structure, run-time type information (RTTI) and object persistency, it has classes for efficiently generating random numbers, various probability density functions, statistical evaluation and pure event driven simulation. The random number generator classes that are included, support a wide variety of discrete and continuous probability distributions.

The simulation part of CNCL consists of classes for simulation events, event handlers, event lists and a scheduler. It is providing also a tool for the integration of new classes into the CNCL class hierarchy.

The statistical part of the library, contains evaluation methods as is the simple moments LRE (Limited Relative Error) and the Batch Means. The simulator has the option of using the batch means method to determine the end of the simulation run.

5.3.2 Simulator Implementation

An occurring LV1 trigger event, will actually create two new data structures, one representing the LV2 and one the LV3 part of an event. Each data structure will be time stamped at each point of the RDPM to the EFU path, it passes. The latencies of occurrence of the different simulation events are implemented as sending to the appropriate subsystem delayed events. For instance, the end of the LV2 filter is simulated as sending to the EVM an event with delay equal to the service time of the filter algorithm.

The current implementation of the simulator, is approximately 8,000 lines of C++ code managed with the aid of CVS [Cede93]. The interface between the different simulation classes (event types and their public accessible methods) is kept relatively simple and organized in such a way, such that any new additional development of a simulation class doesn’t affect the rest of the classes. The individual objects of the RDPM, the switch and the EFU, can be initialized to function in either simple mode or more complex if it is implemented.

5.3.3 Output Data Format

The different performance quantities of the simulated system that are desired as the output of the simulation, are declared at the beginning of the simulation and stored during the run. The choice of the data representation is based on Histoscope [Ferm97], a histogramming tool developed at Fermi National Accelerator Laboratory. Histoscope is a rather simple but powerful data-series histogramming tool that easily integrates with other applications. It provides several plotting options for the representation of the output data. It has a very useful client-server monitoring capability through a simple API. At the simulator initialization, the performance variables, are declared and initialized. During the simulation runs, when new values of those variables are becoming available, they will be plotted and stored in the Histoscope data structures. At the end of the simulation, the values of the individual variables are stored into a file for later analysis.

Using a tool like Histoscope for data monitoring, we are able to monitor closely the evolution of the system until it reaches a steady state. The explanation of the transient phase of the different values monitored, can be very helpful in understanding better and eventually debugging the simulated system.

As the average values and the distribution of the output values can be strongly affected by the transient phase, we have included in the simulator the possibility of transient removal.

5.3.4 Model Validation and Verification

Operational laws like the LV2 and LV3 traffic intensity dependence on the arrival rate and service rate are used for the verification of the simulated systems results. The same laws are useful in the choice of the initial values of the simulation parameters, in order to have a stable system. As the level of modeling of the DAQ subsystems is kept relatively rough, any precision requirements of the output values of the simulation will be limited.

By using simple input parameter distributions (*i.e.*, removing any stochastic effects that might appear) and capturing the behavior of the simulated system at various places, the different output variables are expected to be predictable and hence it must be possible to explain.

Other ways of validating the results obtained by the simulator include selective message tracing and step-by-step execution of messages. Although these are ways that can produce conclusive knowledge of a possible unexpected behavior of the simulator, they are very hard to use due to the amount of information produced. Their use was limited to a few cases that was not possible to understand otherwise.

5.4 Simulation Runs

We will demonstrate the operation of the simulator, in particular the behavior of each of the simulated modules in two considered EFU scheduling cases. In the first case, the results of a simulated system consisting of EFU that favors LV2-trigger jobs over LV3-trigger jobs in the EFU are presented and analyzed. In the second case, we simulate a system that its EFU implement the LV3 preemption for the same set of configuration parameters of the first case.

Input Parameter		**Value**
LV1 Interarrival time	τ_{LV1}	*10 µs*
LV2-filter service time	s_2	*9 ms*
LV3-filter service time	s_3	*900 ms*
LV2-filter rejection factor	R_2	*30*

Table 5.3: Variable simulation input parameters

Input Parameter		Value
LV3-filter rejection factor	R_3	*20*
Number of farm nodes	*nbEFU*	*1,000*
Processors per EFU	*nbCPU*	*5*

Table 5.3: Variable simulation input parameters

The input parameters that are used in both simulation runs, are shown in Table 5.1 and Table 5.3.

5.4.1 RDPM Behavior

The state of the LV2 and LV3 type RDPM can be examined by the occupancy of their Pointer Buffer Tables (PBT in Figure 2.2). When more memory is used in the RDPM, the number of PBT entries in use will be higher. We can obtain the PBT occupancy by plotting the PBT size each time an event fragment enters or leaves the LV2 and LV3 RDPM. Figure 5.6 shows the PBT occupancy of the LV2 and LV3 RDPM for the two simulated systems.

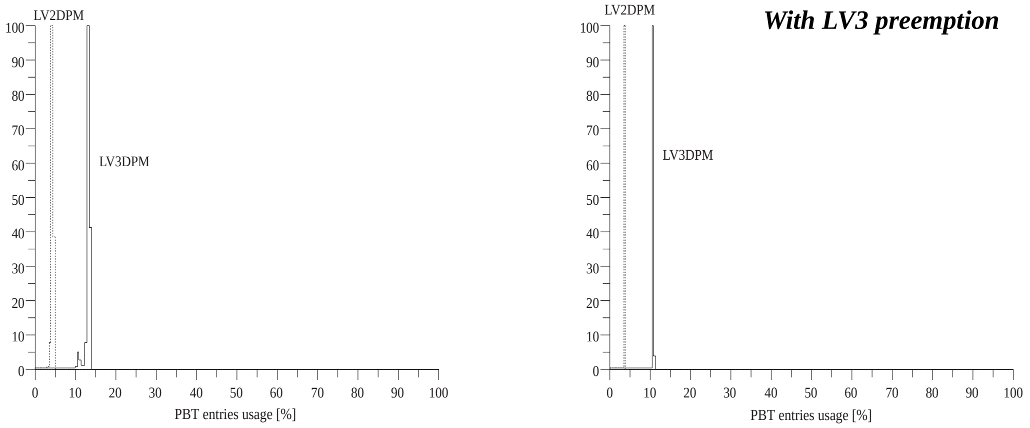

Figure 5.6: Pointer Buffer Table Usage

We observe that more PBT entries are needed in the LV3 RDPM than in the LV2. As can be seen from the figure, approximately three times more PBT entries are required in a LV3 RDPM, compared to a LV2 RDPM. This result matches the expected value which is equal to the ratio of the LV3 event part over the LV2 part *i.e.,* 750KB/250KB when both LV2 and LV3 RDPM have the same memory page size.

The LV3 histogram entries with values lower than the peak, is the transient part of the histogram. This transient can be explained if we take into account that the number of PBT entries during the start-up of the simulation will increase as more LV3 jobs will arrive, until a steady state is reached (that of the peak). Because of the intermediate LV2-trigger processing, stability of the LV3 RDPM occurs slightly later compared to the LV2 RDPM.

We notice also when LV3 can be preempted, the PBT entries required for both LV2 and LV3 are slightly less than in the non-preempted LV3 case. This is an indication that the throughput of the simulated system increases when LV3 is preempted. The throughput increase for the LV2 is obviously because LV2 events will be serviced faster when LV3 is preempted. One the other hand, servicing faster LV2 jobs will result is less waiting time in the LV3 RDPM, before the event is sent for LV3 processing or cleared from the RDPM memory.

5.4.2 Event Manager

The characteristics of the EVM can be captured by the arrival rate of LV2 and LV3 requests from the event filter units and the average number of queued free requests (number of entries in its FRq queue) of the farm.

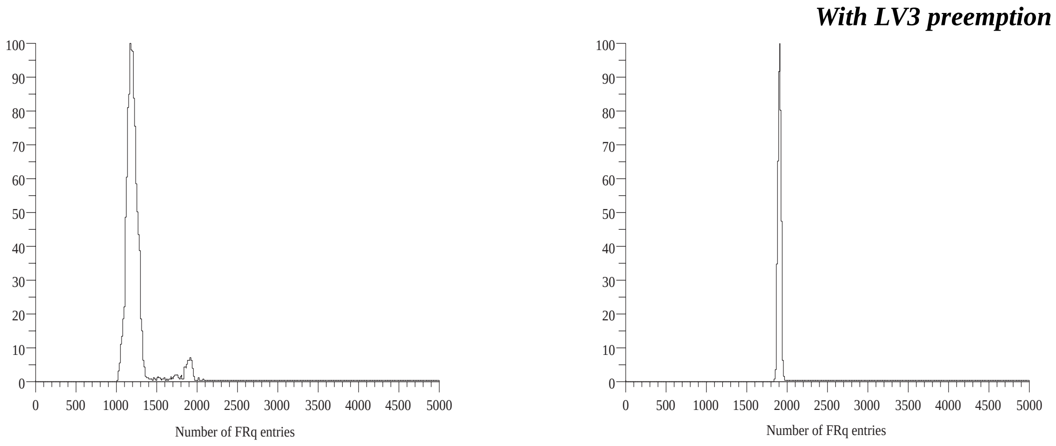

Figure 5.7: Farm Request (FRQ) length

With a stable system, we expected the FRq to have a stable size, less than the total number of FRS (*i.e.*, the number of EFU multiplied by the number of the SFI buffers). However, the average size of FRq not necessarily reflects the number of idle processors. The farm resources communicated to the EVM are considered the SFI buffers of the EFU. The model of the SFI and EFU, considers an SFI buffer free when either a LV2 event-part has been serviced or a LV3 event-part has been received. In other words, having significant amount of entries in the FRq is not alone a sufficient condition for a not loaded farm. It is also the average number of the queued jobs in the EFU that is important, as it will be discussed later. Such a situation of many entries in FRq may occur if the number of buffers in the SFI is set to be larger than the number of processors in the corresponding EFU.

Figure 5.7, shows that when LV3 can be preempted, FRq contains more entries. This is an expected result, again related to the increased throughput of LV2 jobs.

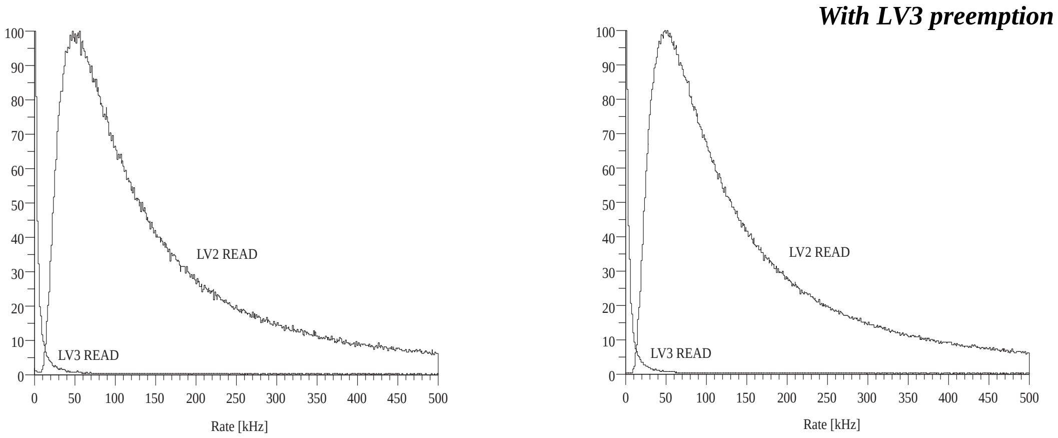

Figure 5.8: Arrival rate of LV2-read and LV3-read requests

Estimates of the arrival rate of requests to the EVM, are also useful to know. These rates are indicative for the bandwidth and latency requirements of the network connection between the EFU nodes and the EVM. The simulator obtains these rates by plotting the elapsed time between successive arrivals of LV2-read and LV3-read requests in the EVM. Figure 5.8, shows the instantaneous distributions of the LV2-read and the LV3-read rates for the two simulated systems. There is no apparent difference between the two simulated system. This is because those two rates depend on the service times of LV2 and LV3 filters as well as on the total number of processors, which are the same for the two simulated systems.

5.4.3 SFI and EFU

The SFI model, assumes a system which tries to maximize the utilization of the SFI buffers. SFI buffers hold either LV2 or LV3 event parts. After the event assembly of a LV2 part the respective SFI buffer is kept busy until LV2 processing has finished. However, after the event assembly of LV3 event part, the SFI buffer is marked free and a request to the EVM for a new LV2 event is immediately sent. In Figure 5.9 is depicted the occupancy of the SFI buffers (*i.e.,* how many of them were on average busy during a simulation run) for the two simulated systems. The data of the two figures have been obtained by plotting the number of busy buffers every time a buffer was marked as busy. The two plots show that indeed, the number of busy SFI buffers tends to be high, as was expected from SFI buffer management. Simulation runs with more buffers per SFI, have shown also high buffer occupancies.

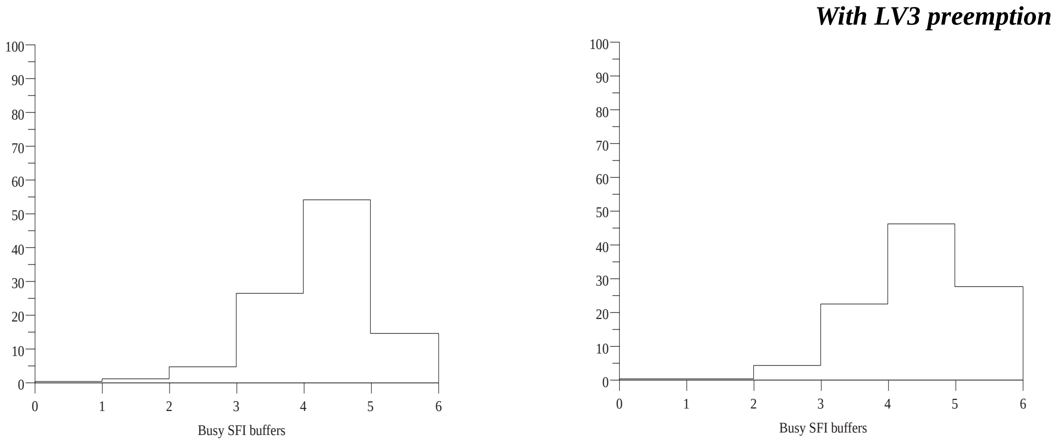

Figure 5.9: SFI buffer occupancy

As it is assumed in the simulation model, event buffering occurs at both the RDPM and the SFI. Therefore, the stability of the simulated system will not be determined only by the RDPM memory size or the number of SFI buffers, but also from the size (or more precisely the growth behavior) of the LV2q and LV3q queues in the EFU.

By plotting the number of entries already present in the LV2q or LV3q when a new job arrives, we can get the distribution of the number of entries in the LV2q and LV3q respectively. Figure 5.10 and Figure 5.11 show the probability that each of LV2q and LV3q has a certain size.

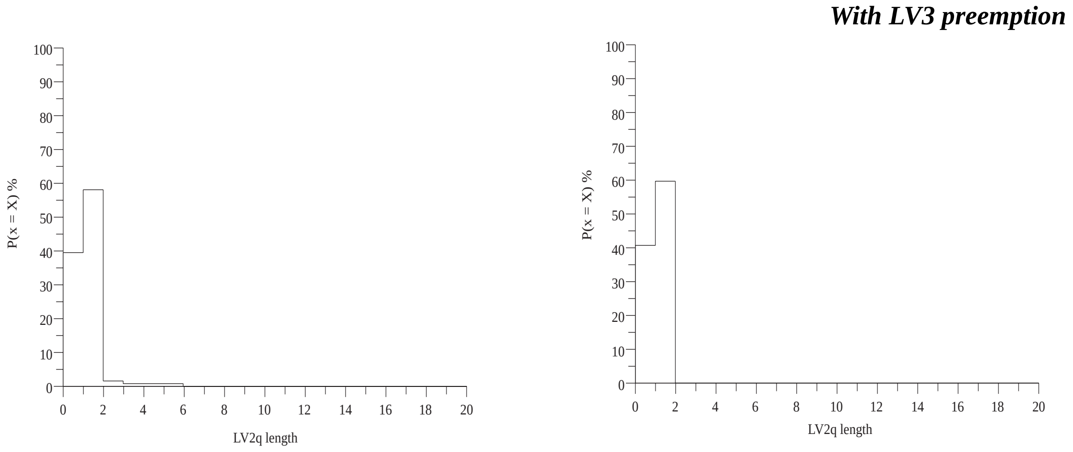

Figure 5.10: LV2q queue length

The two queues in both simulation cases are bounded (there are no long tails), indicating a stable system. Additionally, the LV2q maximum observed size in the preemtable LV3 case, is smaller than in the non-preemtable LV3 case. This behavior of the LV2q is the main reason to have the LV3 filter preempted. A short LV2q (together with a limited s_2) will result in limited LV2 decision latency and consequently lower LV3-RDPM occupancy.

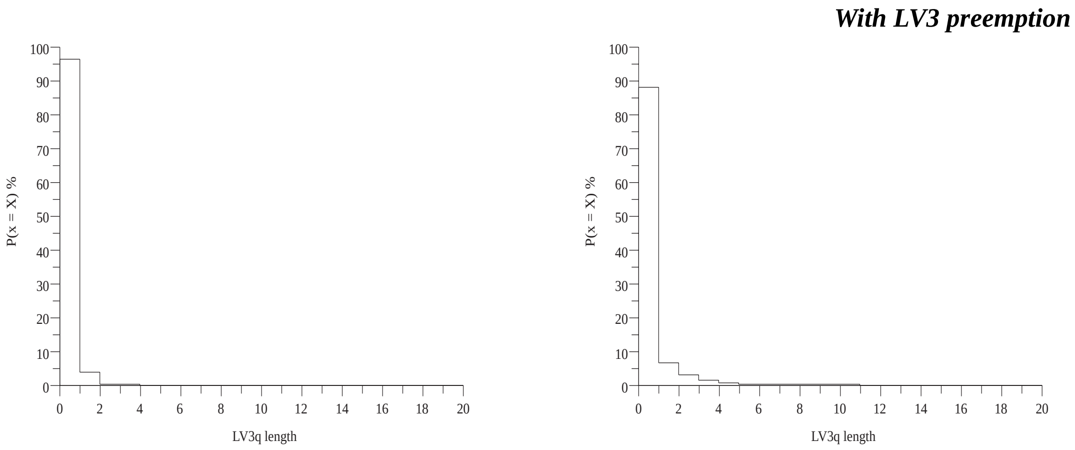

Figure 5.11: LV3q queue length

As a result of the preemption, the LV3q maximum size increases, resulting in increased LV3 decision latency. This is not a problem in a stable farm, as it is only the farm output that depends on the LV3 decision latency. However, the relatively longer tail of the LV3q size when LV3 is preempted, indicates that further steps might need to be taken to confine how many times a LV3 job can be interrupted, according to Formula 5.1.

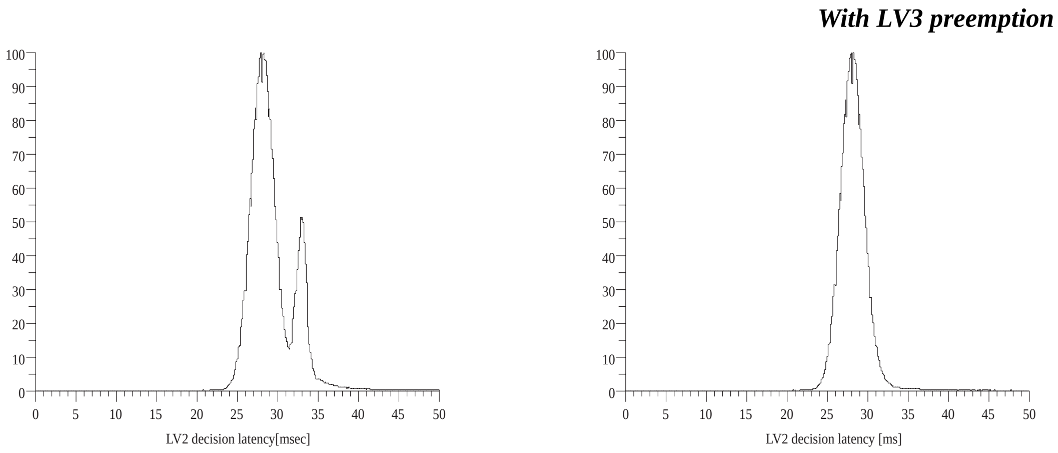

Figure 5.12: LV2 decision latency

The effects of the LV3 preemption are also demonstrated in the distribution of the LV2 decision latency. The LV2 decision latency (Figure 5.12) is defined to be the time interval between the occurrence of a LV1 trigger and the time of arrival at the EVM of the LV2 filter decision.

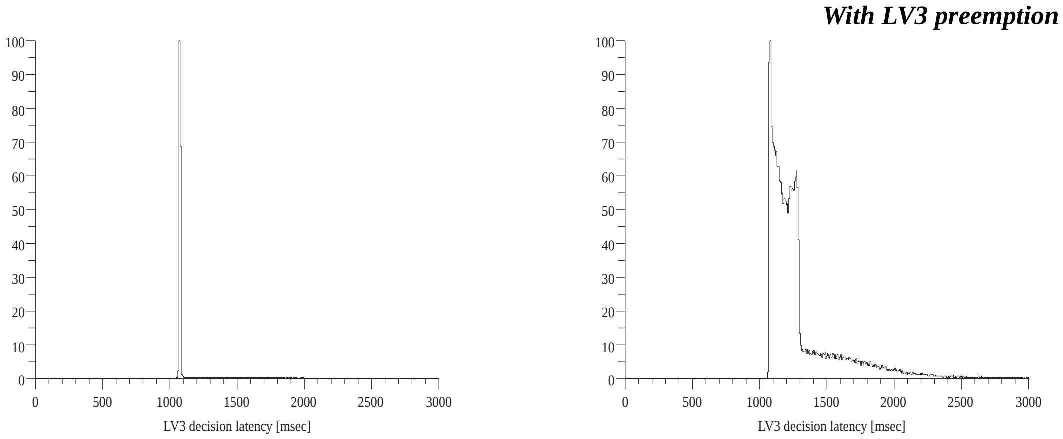

Figure 5.13: LV3 decision latency

Both plots in Figure 5.12 have a peak at ~ 28ms, which corresponds to the sum of the s_2, the average switch latency, the DMA transfer time and the eventual waiting time in LV2q. The plot of the non-preemtable LV3 case, has an additional peak at ~ 33ms. This is explained as the waiting time in LV2q when more than one LV2 jobs are queued.

The LV3 decision latency for the two simulated cases is depicted in Figure 5.13. The peak of the non-preemtable case is much sharper, as the s_3 is much higher from the average switch latency. The effect of the waiting time in LV3q is the tail at the right of the plot. When LV3 is preempted, the LV3 decision latency increases, too.

The shape of the LV3 decision latency distribution is related to the average number of interrupts occurred during the execution of a LV3 job.

We can deduce the average number of LV3 interrupts, by plotting how many times a LV3 job was interrupted at the end of its execution. This is depicted in Figure 5.14.

The number of 25 interrupts on average per LV3 job, as shown in Figure 5.14, is consistent with that obtained by Formula 5.1, which is less than 34.

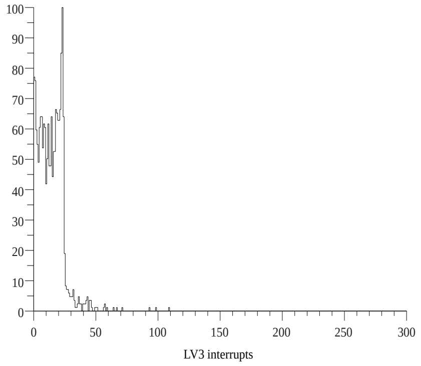

Figure 5.14: Number of interrupts per LV3 job

The overall utilization of the farm is shown in the time-series plot of Figure 5.15. This plot is obtained by sampling the number of LV2 and LV3 jobs running in all the EFU. The vertical axis of each plot is the percentage of the total number of processors occupied by a LV2 or LV3 job, and the horizontal axis is the number of the sample during the simulation run.

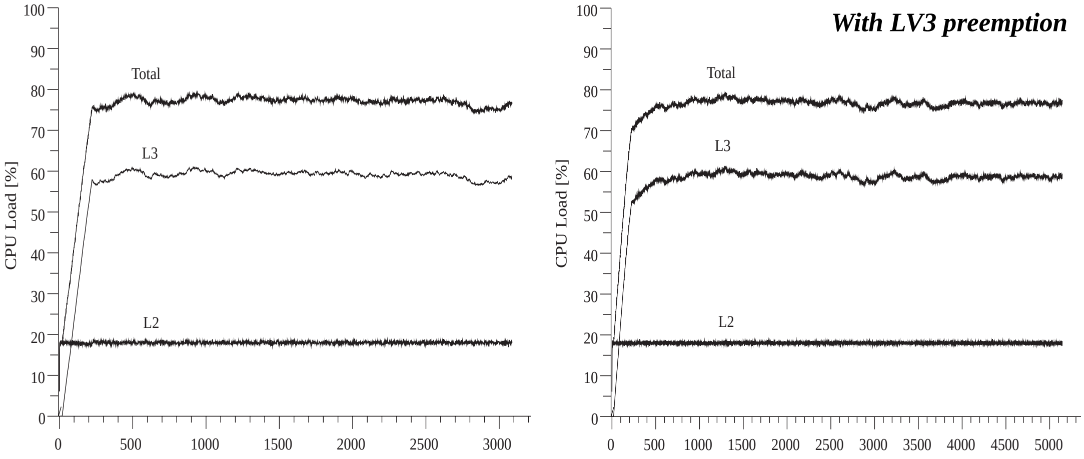

Figure 5.15: Farm CPU utilization

The steep part at the left of each plot, is the transient phase of the simulation, and as it is expected, the LV2 transient is steeper than the LV3. This is because of the time needed for a sufficient number of LV2 jobs to complete and eventually generate a LV3 job before the number of LV3 jobs becomes balanced. If the rejection factor of the LV2 filters R_2 is increased, this transient time is increased also.

The average utilization values of the farm in the two simulated cases are not different. The total load of the farm is approximately 77%. In particular, the load generated by the LV2 jobs is around 18% which matches well what we could predict by multiplying the

LV1 trigger arrival rate with the average LV2 service time. The same is true for the LV3 jobs' load which is approximately 59%.

The instantaneous rate of jobs finished the LV3 processing and were accepted (with a rejection factor R_3) are plotted in Figure 5.16. Their shape is that of a Poisson distributions. When LV3 is preempted, the instantaneous output rate becomes lower. This result can be explained in terms of the increased time interval between successive jobs, passed the LV3 trigger.

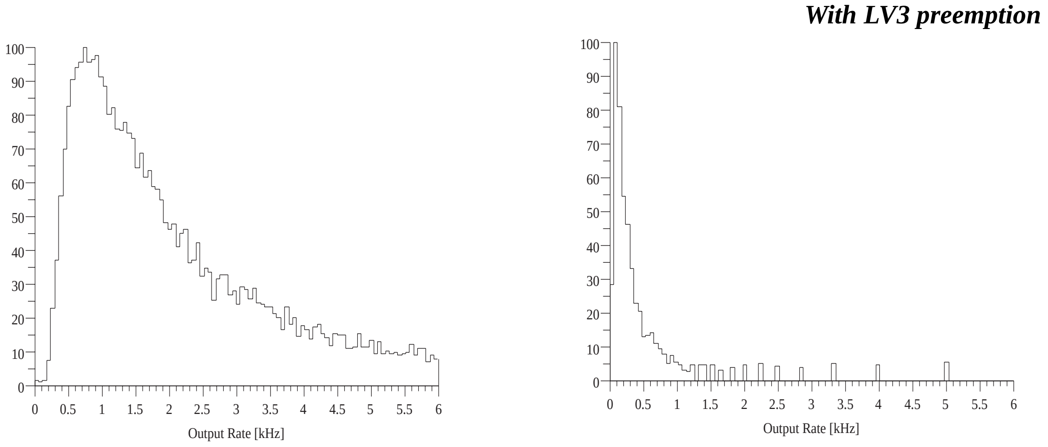

Figure 5.16: Farm instantaneous output rate

5.5 Summary

In this chapter it was described a simulation tool for the CMS event filter farm. It is based on a discrete-time event-driven model of the individual units of the DAQ system. The simulator provides sufficient flexibility for modeling the individual DAQ units with various levels of detail.

This simulator is aimed to assist in the design phase of the event filter farm, by studying different EFU configurations and farm node scheduling policies. It provides several measures in order to evaluate the EFF performance, amongst them are the overall CPU utilization, the decision latencies of LV2 and LV3 jobs and the rate of messages arriving to the EVM.

With that tool, we have performed a simulation study of the impact of preempted LV3 jobs by LV2 jobs. The performed simulations, show that it is appropriate to have the LV3 filters preemtable because as it was discussed in 5.1.1 decreases the LV2 decision latency and therefore increase the system throughput for the critical LV2 jobs. The LV3 preemption option is expected to be even more useful in increasing system throughput when the farm load is close to its maximum. It could be also mentioned that it seems appropriate for the EFU to batch their requests to the EVM in order to reduce the high rate of farm requests arriving at the EVM. This strongly depends on the type of the network connection between the EFU and the EVM and the capability of the EVM to handle high rates.

6 EFU Prototype Environment

In this chapter we look into a prototype setup that emulates the EFU operating environment utilizing a high performance SMP workstation connected to an ATM OC-3 network. Despite the fact that the performances of the individual components that can be used today are by far inferior to those required by CMS, we can study the problems that appear when commercial state-of-the-art SMP systems are used for the EFU.

In this prototype setup we study the performance of a software emulated SFI. Our objective is to prove that already with today's technology, the required SFI performance can be achieved by an emulated SFI in an SMP EFU. The performances of the ATM network, the network controller and the SFI emulator itself will be individually evaluated.

6.1 SFI Emulation

The possibility to build an a SFI emulator arises from the observation that the functionality of the SFI is mostly a memory management problem. That is, the SFI must be able to manage all fragments of each different event, transmitted by the RDPM in an unordered manner. The task of assembling event fragments into one event entity, can be trivially achieved in software by simple memory copying into previously allocated event buffers. Although this may not be the most performing solution, it provides an initial framework, sufficiently flexible for experimenting.

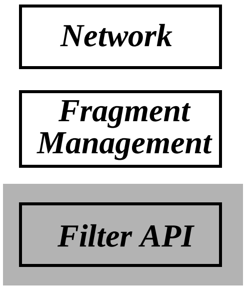

Figure 6.1: Model of the SFI emulator

We model the functionality of the SFI emulator in three layers (Figure 6.1). A network layer to provide the interface to the networking device, a fragment management layer where the fragment assembly takes place, and an API layer to provide the necessary interface to the filtering application running at the EFU.

This decomposition of the SFI emulator, summarizes the independent areas of its functionality. There are no strict boundaries between the three layers, because the implementation of each layer is not entirely independent from the rest of the layers.

6.1.1 Network layer

The network layer provides the interface to the network interface controller (NIC) and the link to the fragment management layer. Depending on the implementation of the SFI emulator, the network layer comprises functions to receive the incoming data and communicate the free buffer resources to the NIC.

The interface to the NIC can be implemented by utilizing standard OS system calls to the device driver. The OS system calls are usually costly in time and therefore may limit the receiving performance of the emulator. Also, depending on the implementations of the NIC driver and the OS kernel buffer management, data may need to be copied multiple times before they are finally delivered to the emulator's buffers. It is obvious that unnecessary data copying will additionally degrade the performance of the emulator. However, using the OS provided system calls the emulator portability is ensured, as these calls are in most of the cases part of standards as POSIX of PASC/IEEE.

A user level driver can avoid entering the kernel of the OS at least for the data transfers and also reduce the number of data copies during the transfers. Depending on the hardware of the NIC, it may be possible to achieve even true zero copying *i.e.,* the data packets are directly delivered to the application's buffers. These performance advantages of user level drivers make them very attractive solutions. However, they are more complex to implement, imply deep knowledge of the NIC internals, and are very much dependent on the actual OS and the NIC they do interface. The complexity may be additionally increased if the cross-domain protection mechanisms required in a multiuser environment have to be taken into account.

6.1.2 Fragment Management Layer

In the fragment management layer, the actual assembly of fragments of different incoming events takes place. Every fragment has a header containing the fragment number and the event number it belongs to. This header has to be processed and then the data-part of the fragment to be placed into the memory location allocated for that event. Also the fragment management layer has to manage the buffers of events that are under assembly, the fully assembled events, as well as the free event buffers. It has to keep track of any events under assembly, for which some of their fragments were lost and therefore their assembly will never complete.

The fragment management layer has to previously allocate sufficient number of event buffers to handle the maximum number of simultaneously assembled events plus the number of events being processed by the filtering processes.

In a more advanced implementation of the SFI emulator, the fragment management layer could handle the processing of an event building protocol, if this is implemented. The use of a specialized protocol layer for the event building and event assembly, has the advantages of potentially lossless data transfers and flexibility for transmitting more than one sub-event in a data packet. It can be also a means of detecting when the assembly of a sub-event has completed. However, such protocols will also introduce additional processing overheads that will affect the event assembly performance. Examples of such protocols are given in [Mand94]. For simplicity, the SFI emulator implementation described here does not make use of event building protocols.

6.1.3 Filter API Layer

The filter API layer contains the set of functions necessary to communicate with the filtering processes running in the EFU. This set of functions is required to be independent of the implementation of the SFI emulator and also must hide the emulator's architecture from the filtering processes.

In a full implementation of this layer, it will be required also to interface to an additional layer which will connect through a separate NIC to the EVM. Such functionality obviously is of very high importance for the full operation of the SFI, but as it requires the presence of the EVM it will not be considered.

The set of functions that the API layer is assumed to support, are requests for the LV2 or LV3 sub-events and requests for the buffer release of a processed event.

6.2 Setup Description

The test setup comprises the RDPM and the SFI emulation hosts (Figure 6.2). The RDPM host (RDPMh) is emulating a data source of event fragments and the SFI host (SFIh) is running the SFI emulator.

The RDPMh is used only to provide the necessary event fragment stream to the SFIh and does not emulate the functionality of a complete RDPM.

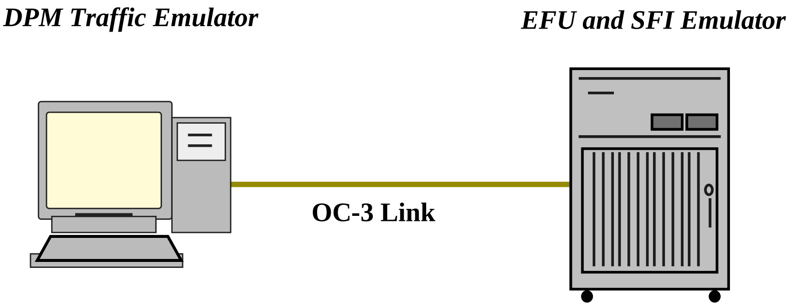

Figure 6.2: Prototype setup

Every node has attached to it an ATM OC-3 (155 Mb/s) Interphase 4615 adapter [Inter94], which may connect the hosts directly (point to point) or through an ATM switch, if more than one RDPMh is to be used.

A SUN SPARC Station 5 workstation with 32 MB of memory and a microSparc processor running at 85 MHz, is used as the RDPMh. A SUN Ultra Enterprise 3000 SMP server with 512 MB of memory and six UltraSparc processors running at 167 Mhz is providing the EFU environment that runs the SFI emulator. Both systems are running Solaris 2.5.1, a proprietary OS based on UNIX System V Release 4. Some relevant performances of the systems used in the setup are summarized in Table 6.1.

	CPU Speed [MHz]	**Memory Bandwidth (Copy operations) [MB/s]**
SPARC Station 5	*85*	*71*
Ultra Enterprise 3000	*167*	*395*

Table 6.1: Performance of the workstations used in the setup

6.2.1 Event Fragment Data Model

In order to emulate a realistic case of the event fragment flow to the EFU, the RDPMh must be able to send an interleaved sequence of randomly ordered fragments belonging to different events. This way, the ability of the fragment layer of the SFI emulator to assemble in parallel multiple LV2 and LV3 sub-events, can be tested.

An event is modeled to have two parts, an LV2 and an LV3, each of them with adjustable size. Every event-part consists of several fragments, like in the real case when many RDPM are used for the data sending. The number of fragments contained in an event-part, can be adjusted. Before an event fragment is transmitted to the ATM network, a header is prepended to it. This header contains the event and fragment sequence numbers, the size of the data included in the fragment and eight additional bytes to be used by the ATM layer of the device driver for the destination assignment. Those eight bytes are an inevitable overhead, required by the device driver itself.

For that purpose, a RDPMh emulator has been developed. It is able to send the fragments of an event in a random order and interleave them with the fragments of *k* other events. The parameter *k* defines the number of events that the SFI emulator will have to assemble in parallel, and it can be modified between different tests.

In order to emulate the data traffic of both LV2 and LV3 fragments, the RDPMh emulator randomly chooses events for which the LV3 part will be also sent to the EFU. The rate of that random choice corresponds to the estimated LV2 trigger acceptance factor. That way, in the flow of the interleaved LV2 fragments are also added LV3 fragments in a similar way to the real case. The exact profile of the fragment flow in a realistic event building system is difficult to determine, as it strongly depends on the traffic shaping function implemented into the RDPM randomizers and the impact of the event building switch.

LV2 Rejection Factor	**LV3-part size [% of event size]**	**LV2 Fragments %**	**LV3 Fragments %**
10	*85*	*63*	*37*
	75	*77*	*23*
	65	*84*	*16*
20	*85*	*77*	*23*
	75	*87*	*13*
	65	*91*	*9*

Table 6.2: Structure of the emulated stream of LV2 and LV3 fragments

In Table 6.2, it is shown the percentage of LV2 and LV3 fragments contained in the data stream generated by the RDPMh emulator. With the gray background are denoted the parameters used in the measurements. Although it seems more realistic to use LV2 rejection rates higher than 20, a worst case was chosen that can give the lower limit of event assembly performance of the emulator.

The RDPMh emulator is designed such that it can run on more than one host, when a switch is used between the RDPMh and the SFIh. In that case, each of the RDPMh will hold only a part of the fragments comprising an event and will send to the SFIh only those fragments. The decision which fragments are owned by each RDPMh is done during the RDPMh start-up, using a unique identifier.

6.2.2 SFI Emulator Implementation

In the design phase of the SFI emulator, it was early recognized that the NIC flexibility available to the network layer of the emulator, may influence significantly the way of assembling events in the fragment management layer. In this implementation of the three layers of the SFI emulator (Figure 6.3) we have adopted the simplest possible solutions mainly because of the restriction to use the software driver supplied with the NIC.

During the emulator initialization, a fixed amount of event buffers is allocated. They are all placed into the Free Event Queue (FEQ). Another event queue is created (ERQ) to hold the buffers of assembled events and it is initialized to be empty. To keep track of the events under assembly, a hash table is also created. This hash table is indexed by the event number of a fragment and it has a sufficiently large number of entries to avoid collisions.

The UNIX System V *streams* mechanism [Ritc84] was used in the network layer, to receive the incoming data stream from the NIC. Streams provide a bidirectional data and control path between the NIC device driver and the user application and they are very well suited to accommodate network protocol stacks. Most parts of the communications of the used OS are based on streams.

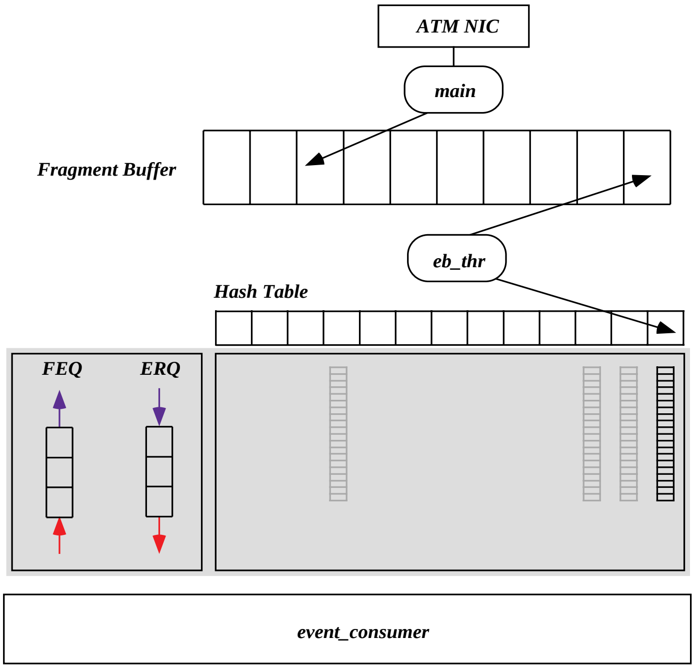

Figure 6.3: Architecture of the SFI emulator

A streams-based device driver was the only option available from the manufacturer of the NIC. For that reason, the main thread of execution of the emulator is implementing a fast loop of `getmsg()` calls to the file descriptor of the NIC device driver in order to receive the data. The incoming data are placed at the tail of a circular buffer, denoted as *Fragment Buffer* in Figure 6.3.

At the initialization of the emulator, a second thread (eb_thr in Figure 6.3) is created. The *eb_thr* will detect a non-empty *Fragment Buffer* condition and consequently will start to retrieve entries from it. For every fragment that eb_thr processes, it first compares its event number with that pointed by the hash index. If no other fragments of this event have been received, it will retrieve a new event entry from the FEQ and place it into the hash table. If the event number of the fragment and the one in the hash index match, it will copy the contents of the fragment to the appropriate memory location and increase the event's received fragments counters. If a fragment is the last one of an event, it removes it from the hash table and places it into the ERQ. Without the use of higher level protocols, a sub-event is defined to be completed when a predefined number of fragments has been received.

To emulate the event filtering taking place in the EFU, one or more additional threads are started (event_consumer in Figure 6.3). The event_consumer threads will be notified once the ERQ is not empty, to process incoming assembled events. The notification is done through conditional variables and mutexes placed around the ERQ access calls to ensure the data integrity of the ERQ. The same functionality of the event_consumer threads can also be achieved if a separate process is started.

In that implementation, the event processing functionality is substituted by a simple delay of a specified amount of time. Once this delay time has elapsed, the event is placed back into the FEQ and the next available event from the ERQ is retrieved. In order to have a more realistic estimate on the synchronization overheads involved, the FEQ, ERQ and all the event buffers are placed into a shared memory segment, marked with the gray background in Figure 6.3. That way, independent of the use of threads or separate process for the event_consumer function, the synchronization overheads are equally expensive.

6.3 ATM Performance

Before evaluating the performance of the SFI emulator, a series of measurements of the data transfer throughput and receive latency in a point to point ATM network were performed. The setup was the same as that in Figure 6.2. The application to application bandwidth was measured for different sizes of data packets, using the same set of transmit and receive system calls, as in the SFI emulator.

For every measurement, the transmitting host was sending 200,000 AAL5 data packets. The time to receive all the packets at the receiving end was measured. The AAL5 layer can encapsulate in an AAL5 packet, data packets (PDU) of size up to 64 KB [Hein93]. A trailer is appended to the PDU containing a CRC-32 checksum and a padding part to make the total size of the packet multiple of an ATM cell size. The AAL5 packet is segmented into ATM cells of size 53 bytes before it is send to the network. The layout of an AAL5 packet is shown in Figure 6.4 and the segmentation into ATM cells in Figure 6.5.

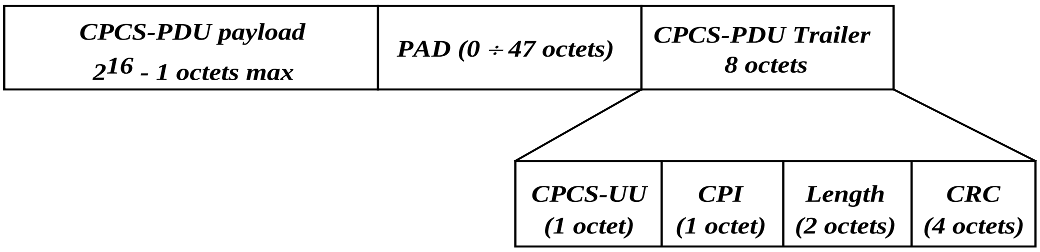

Figure 6.4: AAL5 packet layout

It has to be noted that in all measurements, no packets were lost in the network and none dropped by the hosts. Figure 6.6 shows the theoretical, the maximum possible and the measured throughput, for the OC-3 type of ATM links. The theoretical curve is an expression of the bandwidth that an application is expected to receive, when all ATM-cell and AAL5 related overheads are substracted from the maximum possible bandwidth of the physical layer.

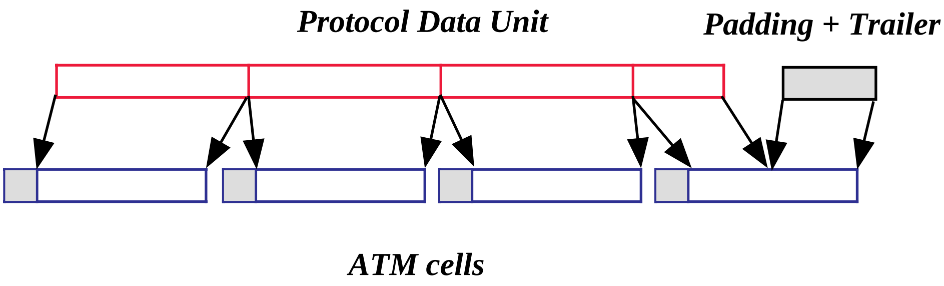

Figure 6.5: AAL5 segmentation into ATM cells

From the theoretical performance curve, it is seen that already for packets larger than 500 bytes, the bandwidth is very close to the maximum value (134 Mb/s). The saw-tooth shape of the theoretical curve is due to the effect of the padding trailer in AAL5 packets. The padding can be as large as almost an ATM cell payload, and as it is expected, it has a higher impact on the measurements where smaller packet sizes are used.

The measured throughput however, is much lower than the theoretically predicted. In particular, for packet sizes of 1 KB, the measured value was around 33 Mb/s, while it was expected to exceed 100 Mb/s. It has a smooth shape because the measurements were performed with packet sizes of powers of two, which gives a constant padding length of 24 bytes.

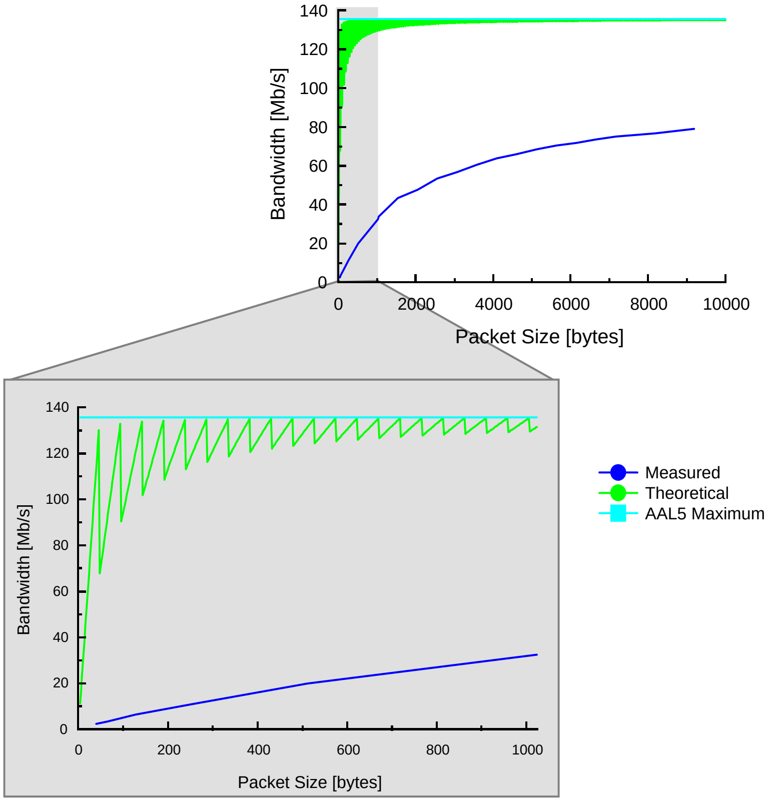

Figure 6.6: ATM AAL5 Theoretical and measured performance

Measurements with packet sizes larger than 9188 bytes could not be performed. The NIC has been optimized to operate in a TCP/IP network and the maximum packet value built-in the NIC design, corresponds to the default MTU for IP over ATM [Atki94]. Similar point-to-point ATM measurements performed over TCP/IP, have not shown different results for the obtained bandwidth [Fouquet]. Answers to the question where the performance is getting lost, must be inquired in the NIC itself, the host hardware and software, or in a combination of both [Drus93].

6.3.1 NIC Performance

The NIC used in the measurements, implements a two stage transmission scheme. The data packet to be transmitted, is first copied from the OS kernel buffers into the device driver's buffers. From there it is copied to the NIC adapter buffers using a DMA transfer, where the AAL5 packet is formed and segmented into ATM cells before it is transmitted to the network. In that scheme, the NIC must issue at least two interrupts, one when the DMA has finished and the NIC front-end must be programmed to inject the data into the network and a second when the transmission has finished and the allocated resources have to be freed. It is designed however in such a way, that both interrupts are entirely avoided, except in the case of an error.

When the driver receives data, the opposite to the transmit sequence of events is taking place. The incoming packets are first copied into the NIC buffers, then transferred through DMA to the NIC device driver buffers and finally are copied into the OS kernel streams buffer. The last copy may be avoided by loaning the driver's buffers to the streams sub-system. Buffers holding packets of size larger than 2 KB are loaned using the extended streams buffer mechanism (`esballoc()`). When the NIC receives data from the network, it generates two interrupts for every AAL5 packet. One interrupt when the packet is in the on-board memory and a second one when the packet is received into the system memory.

In order to study the performance of the NIC and its associated driver, we can model the time it takes for a data transfer to and from the network, using two distinguished types of overheads. A *constant overhead*, referring to the overhead that does not depend on the size of the transferred packets, and a *variable overhead* that is proportional to the packet size.

The constant overhead is due to operations that have to take place in each data transfer (for both receive and transmit) and result in a determined time delay. Typically, constant overheads are determined by the number of interrupts required per data packet, the number of slave accesses to the NIC from the device driver and the protocol header processing time. Usually, interrupt handlers do not have processing time that is dependent on the packet size. The slave accesses are performed in order to program the NIC registers before and after the data is transferred and also to obtain NIC status information. The device driver that was used in the setup, performs a relatively high number of slave accesses per data transfer. To reduce the slave accesses a local copy of the registers whose state does not change often it is cached in the device driver. Even with this caching mechanism, the number of slave accesses remains high.

The variable overhead is due to the DMA speed of the transfers between the NIC and the memory sub-system, and the number of copies that are performed for each packet. The segmentation, reassembly and checksumming times of AAL5 packets, could be also adding to the variable overhead. They are carried out during the time the packet is injected to the network by specialized ASIC in the NIC and therefore have no significant contribution to the variable overhead.

This performance model is a simple way to evaluate the performance of an I/O device. Its simplicity is its advantage but also its drawback. It is not a precise description of what takes place during a data transfer between the NIC and the user application. Therefore, it cannot provide sufficient insight on the timing of each of the tasks needed for a data transfer. Despite that drawback it is useful to understand which are the limitations of the studied setup. More sophisticated models, for example LogP [Cull93], can provide more information, but in a much wider scope including the network, machine and the communications application.

Using the above described model of the data transfers between the NIC and the host system, we can evaluate the NIC performance by measuring the time it takes to receive a series of packets of various data sizes. We can then fit the obtained data with a straight line $y(x) = a + bx$, the coefficients of which correspond to the transfer latency (a) and the inverse of the effective transfer speed (b).

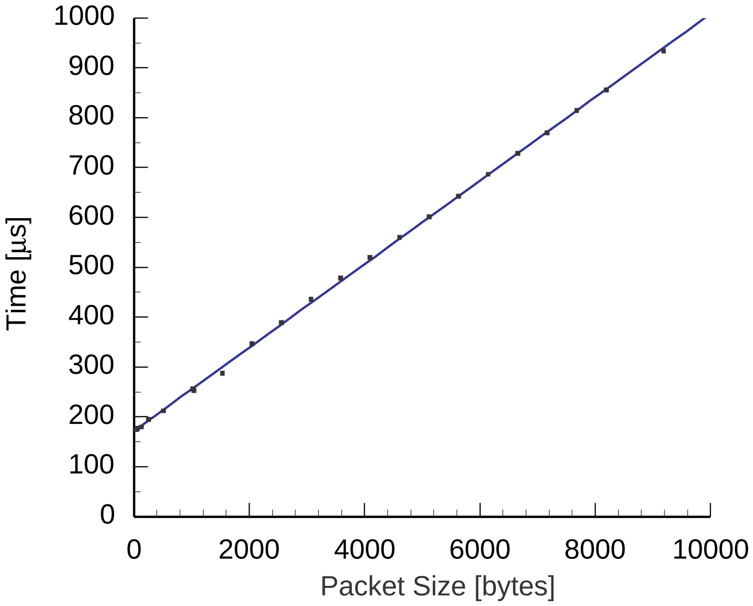

Figure 6.7: Packet receive latency

Figure 6.7 shows the measured latency of packet transfers with different sizes. The constant overhead is approximately 170 µs. This is a relatively high value and as it is shown in Figure 6.6, it makes small packets extremely costly. For comparison, we can mention that sophisticated drivers made specifically to reduce to a minimum the transfer latency have a constant overhead below 20 µs [Wels97]. The cause of such a high overhead is the number of the slave accesses to the NIC during the DMA setup cycle and the two interrupts that take place in each transfer.

The effective data transfer speed as it is obtained by the linear fit is around 12MB/s. This is a very low value for the bandwidth of the data transfers between the NIC and a user application. It is well below of the DMA transfer speed which is expected to be in the order of 60 MB/s (32 byte I/O bus at 20 MHz). Contrary to the constant overhead, it is more difficult to explain the breakdown of the variable overhead, because of the OS influence. The measurement of the transfer time is influenced by the process scheduling and the memory (buffer) management of the OS. The above figure of the effective bandwidth between the NIC and the application, shows clearly that traditional I/O interfaces are not always suitable for high performance I/O transfers.

6.3.2 Host Hardware and Software Performance

The implementation of the SFI emulator relies on sufficient memory bandwidth to be available on the host running as an EFU, because of the necessary fragment copy. The memory bandwidth available on the host must also be sufficient to accommodate the eventual needs of the filtering algorithms and must scale by the number of processors and the active processes in the system.

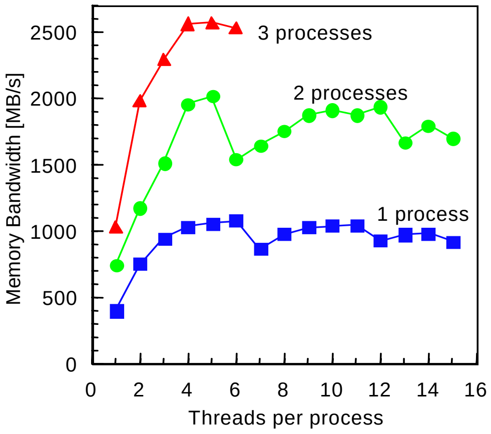

Figure 6.8: Memory bandwidth and scaling

In order to evaluate the above behavior on the studied system, we used a modified version of the STREAMS benchmark [McCa95]. The original benchmark was modified to spawn multiple “copying” threads in one process and eventually start multiple processes. The results of that benchmark when the number of threads per process is varied are shown in Figure 6.8, for one, two and three concurrent benchmarking processes. The single-thread, single-process result is the relevant one to SFI emulator because the last is designed to do the fragment copy in a single thread (eb_thr in Figure 6.3).

It is interesting to note that the machine can sustain an effective aggregate memory bandwidth which is very close to its peek aggregate bandwidth 2.6 GB/s [Char97]. From the results of the benchmark we can conclude that the memory bandwidth (~395 MB/s for the emulator case) is not limiting the performance of the SFI emulator. It is shown that sufficient resources are available not only to the emulator but also to other applications.

The same ATM bandwidth measurements as in 6.3.1 were performed using the faster host (UE-3000) for transmitting and the slower host (SS-5) for receiving. The data throughput improved significantly, but frequent loss of data packets at the receiving end was observed. An analysis of the NIC device driver statistics, revealed that all sent packets were received by the NIC front-end hardware at the destination host. It turned out that it was the OS streams buffer management between the device driver and the network layer of the emulator, that could not cope with the high data rate and for that reason the streams flow control mechanism was activated. Consequently, the NIC device driver was disabling the adapter interrupts, hence dropping incoming packets. This performance problem cannot be attributed to any memory bandwidth limitations of the receiving host, which could be due to excessive packet data copying. The bandwidth required for packet transfers at 155 Mb/s ATM speeds is more than ten times lower than the memory bandwidth available at the receiving host. The most probable cause is the streams buffer management complexity. Despite the fact that streams are not copying the buffer contents, it seems that due to overheads the buffer processing rate is lower than the packet arrival rate.

The above observation suggests that standard OS provided mechanisms of accessing data from network devices like streams, may easily reach their performance limits if they are used with NIC with more demanding network speeds. Otherwise, they may perform well in legacy networking speeds as that of Ethernet.

6.4 SFI Emulator Performance Measurements

From the ATM performance measurements on the test setup, it became clear that the evaluation of the SFI emulator performance can be obscured by the poor network layer performance. For that reason, the emulator was modified to operate in two different modes. In the first mode, the emulator receives fragments from the ATM network and assembles the fragments into events simultaneously. We will call this mode *simultaneous assembly* (SA).

In the second mode, the network layer will first fill the Fragment Buffer (Figure 6.3) and only after that, the fragment assembly thread (eb_thr) will be started. We will call this mode *deferred assembly* (DA) — the event assembly is deferred until the fragments to be assembled have arrived. The performance of the fragment assembly will be measured during the execution of eb_thr and thus will not include any network activity. When the emulator is operating in the DA mode, the size of the Fragment Buffer is increased to accommodate a sufficiently large number of fragments for the different measurements.

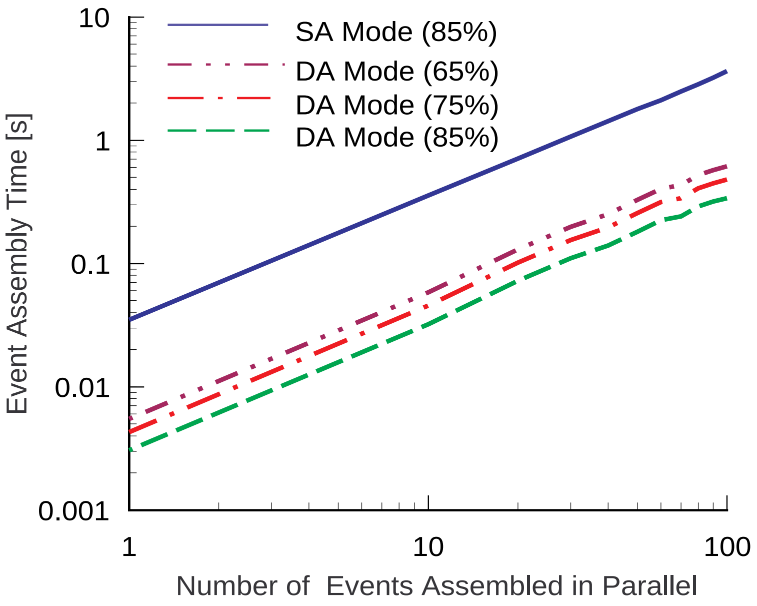

Figure 6.9: Event Assembly Performance

As it is depicted in Figure 6.9, in the SA mode, the event assembly time is entirely limited by the ATM network performance. Indeed, for a sequential (one event at a time) event stream, the event assembly time is corresponding to the 35 Mb/s limit of the ATM network.

Operating the emulator in DA mode, we measure the assembly time of one event for three different ratios of LV3-part sizes over the total event size. As it is expected, the per-event assembly time is much shorter than in the previous measurements of simultaneous receives and assemblies.

In the case of the smallest LV2 event part (15% of the total event size), we observe the maximum event assembly throughput of the emulator, while in the case of the biggest LV2 part (35% of the total event size), the lowest throughput. This is an indication that the throughput is entirely dominated by the speed of the copy operation required for the event assembly. In Table 6.3 is summarized the emulator's performance in the cases of one and 100 parallel events, for the three cases of the LV3 part size. The last column gives the assembly throughput for 100 parallel events.

LV3-part size [% of event size]	LV2 Assembly Time (1 event) [ms]	LV2 Assembly Time (100 parallel events) [ms]	LV2 Assembly Rate (100 parallel events) [Hz]
85	*3.0*	*345*	*289*
75	*4.3*	*485*	*206*
65	*5.5*	*619*	*161*

Table 6.3: Performance of the emulator in DA mode

The emulator’s throughput in all measured cases is well above the 100 Hz threshold, required at each EFU port for an EVB of 1000 × 1000 ports. This is a very encouraging result, as it shows that a rather simple implementation of the SFI in software may perform equally to a separate SFI unit. Evidently the emulator’s performance can be further improved by using more sophisticated fragment assembly algorithms and OS interfacing.

6.4.1 Contention Analysis

In a concurrent application as this simple SFI emulator, the resource contention that occurs in critical code segments like in the fragment layer, may significantly limit the performance. An assessment of whether the synchronization primitives used between the three different threads of the emulator could be a bottleneck, may help to avoid design pitfalls and potentially assist to the improvement of the emulator.

For that purpose, a commercial thread analysis tool THA [Sun95] was used to sample the execution of the SFI emulator and provide performance measures for each thread as CPU execution time, mutex and conditional variable waiting times. THA can sample the execution of a compiler instrumented executable and generate trace output files, which can be later analyzed.

The following contention analysis study is done with the emulator running in the SA mode. If the DA mode was used, the tracing output would have been dominated by overheads generated by the Fragment Buffer locking mutexes, used to avoid underflow of the circular buffer. This is because the fragment assembly runs much faster than the buffer is filled. In the DA mode, all mutexes are still entered, but those related to the Fragment Buffer are not expected to block. Also, the size of the Fragment Buffer is intentionally smaller than that used in the throughput measurements, in order to produce more readable trace output. Otherwise, the scale of the time axis could have been much higher and the shape of the curves smoother, hiding the different phases of emulator’s execution.

In the output produced by THA it is displayed the CPU time, mutex and conditional variable waiting times of the main thread (Figure 6.10) the event assembly thread (Figure 6.11) and the event consumer thread (Figure 6.12). All figures have at their horizontal axis the absolute time since the start of the emulator execution and at the vertical axis the amount of processor time in percentage, for each of the performance metrics. Examining the THA output for the emulator, we can distinguish the following four different phases.

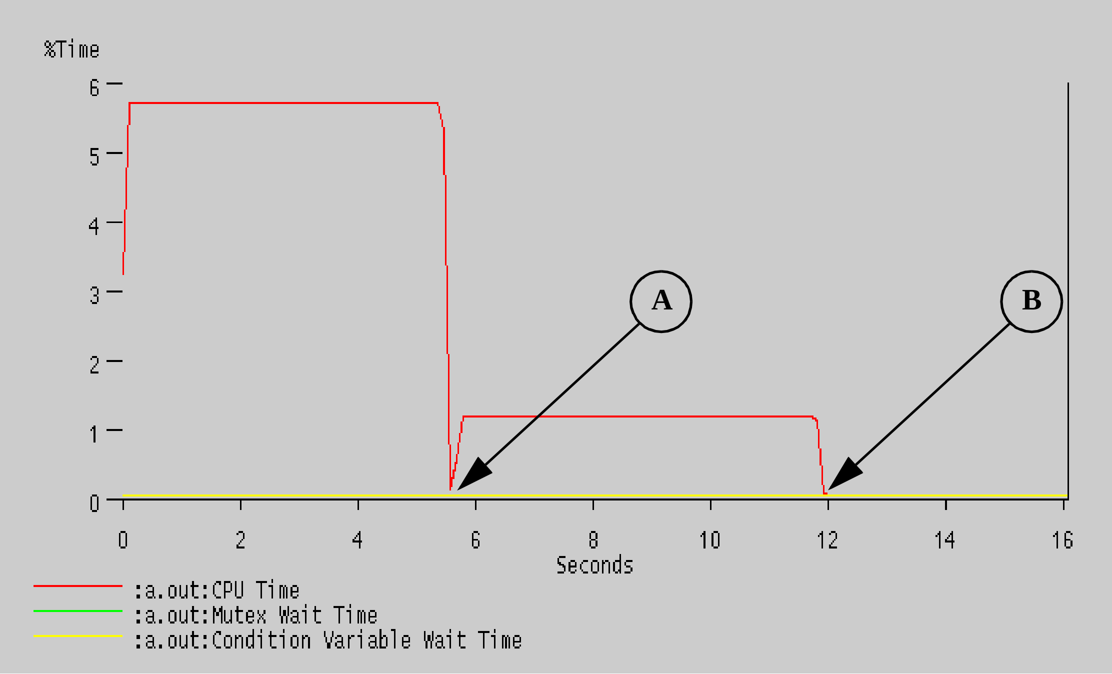

Figure 6.10: Profile of the main thread

Start → A. We have activity only in the main thread as the other two are not yet created. This is the emulator initialization, where all the memory allocation of the resident event buffers takes place. This accounts to a very little (~ 6%) CPU time for almost 5.5 seconds.

A → B. Immediately after, the event consumer thread is created, which in turn suspends execution, waiting for the condition ERQ to become non-empty.

After the main thread creates the event consumer thread, it enters the receiving loop to fill the Fragment Buffer. The filling of the buffer seems to require very little CPU user time, as most of the elapsed time is accounted as system time.

B → C. After the Fragment Buffer is filled, the main thread will continue waiting to be signalled that the rest of the threads have terminated and then exit.

The event assembly thread is created and started. It goes through the Fragment Buffer, pulls entries from the FEQ for every fragment of a new event it encounters or copy the fragment data if it is of an already received event. As it is depicted in Figure 6.11, alternating peaks of CPU execution and mutex waiting, dominate the fragment assembly phase. The alternate nature of the CPU and mutex peaks is due to the way the event assembly is done. For every fragment retrieved from the Fragment Buffer, a synchronized check for a buffer overflow or underflow is performed. The exact number of peaks shown in the same figure, has to do with the sampling granularity of the THA and not with any peculiarity of the event assembly thread.

No events are still fully assembled, the ERQ is still empty and the event consumer thread continues to be suspended.

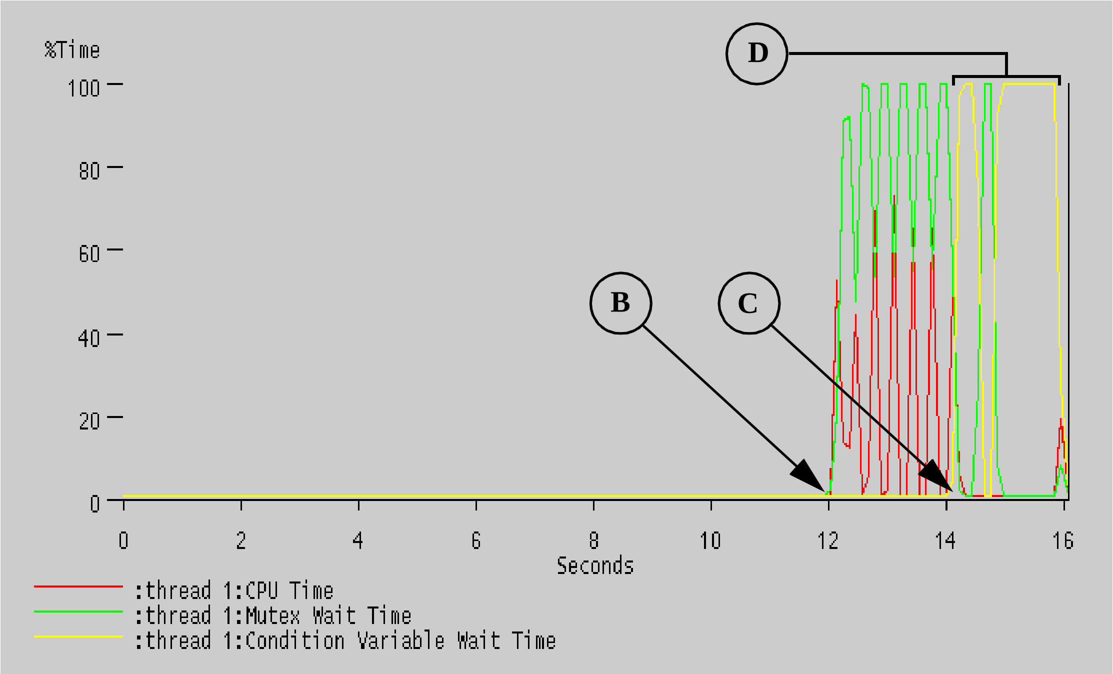

Figure 6.11: Profile of the event building thread

C → End. This is the time when the first assembled events are placed in the ERQ. There is very little left for the rest of the events to become ready and the CPU time of the event assembly thread is reduced. The mutex waiting time is reduced also and the condition variable waiting time is dominant, because the event assembly and consumer threads, synchronize accessing the ERQ and FEQ. This is the phase denoted with the D arrow in Figure 6.11.

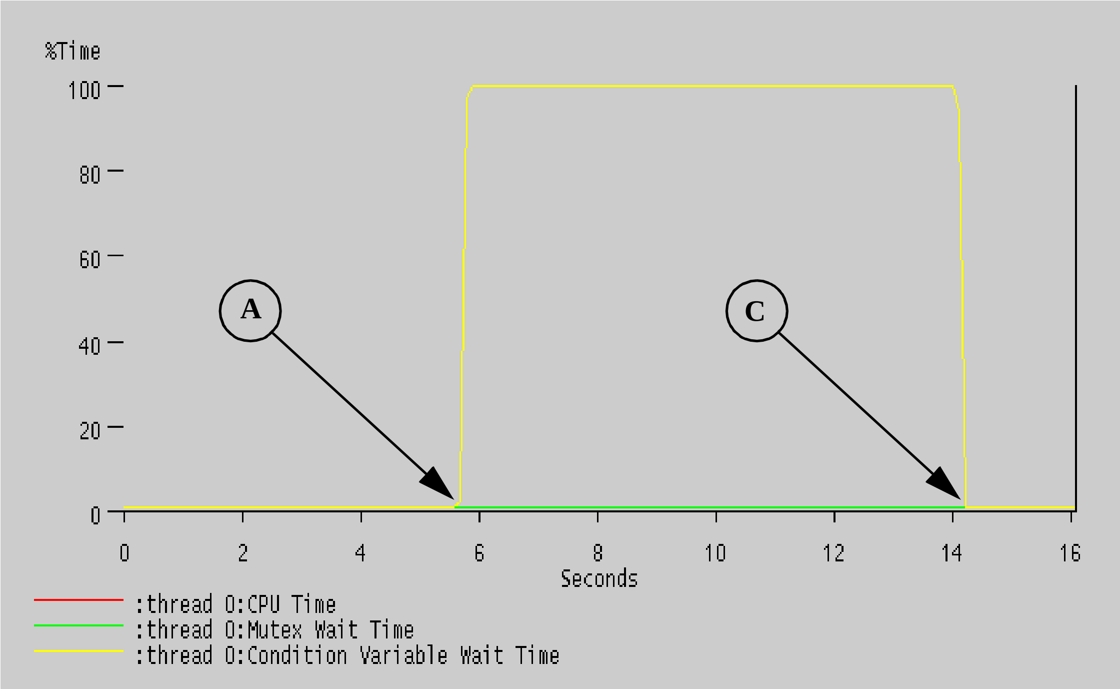

Figure 6.12: Profile of the event consumer thread

The conclusion from that analysis is that thread synchronization with mutexes and condition variables, must be used very carefully. The case of the event consumer and event assembly threads accessing the ERQ and FRQ, points to a potential bottleneck that may occur, especially if it is taken into account that more than one consumer thread (or process) may run in parallel in an EFU.

6.5 Performance Considerations

With this small prototype setup, it became possible to identify most of the potential problematic areas that can appear when off-the-shelf commercial computer systems are used for the EFU and the SFI. It became evident that the main performance burden falls onto the networking subsystem and much less to the specific event assembly problem. A more detailed discussion of these two issues follows.

6.5.1 NIC and Host Related

The performance characteristics of an NIC, can be divided into three categories. Those related to the physical media and network protocol which the adapter is interfacing to (*e.g.*, ATM, FDDI, etc.) those related to the host attachment point at the I/O bus and finally those related to the application interfacing.

Relevant to the first category of performance characteristics are the network's packet size and the packet processing.

In the case of an ATM adapter, the transmitted or received data unit is an ATM cell of 53 bytes. The front-end circuits of an ATM adapter can start to transmit with only a few ATM cells initially available into the NIC buffers. This is relatively a very little amount of data as compared to Ethernet for instance, where the packet size is 1518 bytes and even bigger in the case of FDDI. This property of ATM and in general of cell-based networks, is one of the reasons why ATM has the potential to provide low-latency communications. For the current early generation of ATM NIC however, this is not always the case.

The data segmentation/reassembly into and from cells is a typical function performed by the adapter itself and not by the host processor. Additionally, transport protocol (AAL5) checksumming is also performed by the hardware. Some of the adapters available today, can be instructed also to do protocol processing which may further facilitate the placement of the incoming data into the host's buffers. Such built-in functionality is expected to be very interesting for our event assembly task. It can reduce further the multiple data copying and eventually compensate the additional overheads when adding an event building protocol.

Some ATM NIC available today, like the Fore SBA-200, incorporate an on-board communications processor. In the SBA-200 case, it is mostly used for the fragmentation, reassembly and checksumming of the AAL5 packets, as well as for transferring data to and from the host memory. Adding a processor in the NIC, increases flexibility as in most of the cases it reduces the presence of specialized hardware in the adapter. This flexibility is very useful in the early stages of the NIC prototype and development.

The existence of a communication processors (CP) will off-load the host processor for network transfers. CP are often used to add bus master functionality to the NIC. In such cases, the overheads of data transfers from and to the adapter are much higher and consequently increase the network latency. The SBA-200 board has to make several memory accesses when receiving or sending AAL5 packets in order to access the lists of buffer descriptors in the host memory. Replacing the DMA engine with a CP, may simplify the NIC design, but it may also reduce its performance. Conversely, the presence of a CP instrumented with a DMA engine to access the host's main memory will offer both advantages of host processor off-loading and onboard early processing of packets. One could argue on that point that it is cost-unjustifiable to have both a communication processor and DMA engines available in an adapter.

Another important NIC design issue is the amount and type of buffering available on board. In particular the amount of NIC buffering is often playing an important role in the flow control behavior of various network protocols. The onboard buffering is commonly treated as a trade-off between the adapter's cost and performance. Sufficient on-board buffering, may reduce or even eliminate the need of host buffering and consequently reduce the number of times that data is copied. This can be achieved when DMA transfers are used to move data from the application's to the NIC buffers. An ATM NIC may need lots of memory in order to buffer several full-sized (64 KB) AAL5 packets, because of the maximum AAL5-packet size. The type of the memory used can inhibit the advantages of on-board buffering. Dual-ported memories that allow simultaneous accesses from the NIC front-ends and the I/O bus controllers, will perform best. Otherwise, the serialized memory accesses will limit the adapter's performance. The above problems in designing high performance NIC has been in-depth investigated by many researchers [Davi93] [Drus94] [Stee94].

A key performance issue in small-message, low-latency communications turn to be the relative merits of DMA and PIO transfers. DMA is very efficient in relatively large data transfers (in the order of one KB and higher), however it usually takes several slave accesses to program the DMA engine. It is very often the case that for small packets (less than a few bus burst-sizes *i.e.*, typically less than 100 bytes) PIO is performing better than DMA. The device driver of the NIC must provide the flexibility of using PIO or DMA transfers, depending on the packet size and the transfer performance.

The number of interrupts generated by the NIC hardware during the reception or transmission of data packets is a crucial performance factor. Each generated interrupt implies at least a context switching and usually a processor's cache flushing. Depending on how the OS handles interrupts, in its interrupt context, it may search several interrupt handlers before the NIC interrupt processing is started. When an interrupt occurs, a cache with modified data will need several main memory cycles before it is flushed. The total interrupt processing time times the number of interrupts generated when the NIC is receiving or sending, will determine the maximum packet rate the NIC can handle.

	Controller	**Embedded Processor**	**Full-blown Processor**
I/O Bus	*IPhase 4615* *SUNATM*	*Myrinet* *Fore SBA-200*	
Memory Bus	*TMC CM-5* *Cray T3D*	*Meiko CS-2*	*IBM SP-2* *Intel Paragon*

Table 6.4: NIC Attachment points in commercial computer systems

The question of the NIC attachment point into the host system is frequently raised in several research projects and it is commonly referred to as an important requirement to achieve high performance in networking [Davie]. The general tendency (Table 6.4[1]) is that MPP systems have attached their NIC closer to the memory subsystem, while in the

---

[1] Part of the data are from [Henn90]

workstations and servers the I/O bus is used instead. The memory bus attachment provides lower latency than on an I/O bus. On the other hand, I/O buses are more standard among different computer systems and cheaper adapters can be manufactured for them as they can be targeted to many different systems. The increase of bandwidth requirements and eventually a wider acknowledgment of the importance of communication latency, is expected to lead to the appearance of more NIC attaching to the memory bus.

A rather old technique to minimize the impact of buffer copying, is page re-mapping. It can be particularly useful when is used in hosts that support DMA to virtual memory locations (DVMA). Page re-mapping can be very advantageous if the host system can make the TLB updates (several entries may needed to be updated) and memory locking/ unlocking, faster than copying the data. This is a function of the transferred packet size.

In our prototype it became evident that the OS intervention in the path between the NIC and the SFI emulator, was not favoring data transfer performance. OS have been often found not following the performance of faster hardware [Oust90]. In particular the issue of the OS presence in high performance networking has received significant attention and is commonly acknowledged in gigabit networking research projects [Chio96] [Eick93] that an effort should be made so that data between the NIC and the application are moved with minimal OS intervention. It has been shown that user-level DMA (UDMA) has superior advantages over the traditional ways of making data transfers [Mark97].

Although our setup is not limited by memory bandwidth performance, sufficient resources must exist not only to accommodate the needs of the SFI emulator but also the needs of the filtering applications supposedly running in the EFU. Lower-end computer systems may be particularly prone to insufficient memory bandwidth. The networking performance also strongly depends on the available memory bandwidth [Alme95] [Ande91].

All above issues of NIC hardware and software design, as well as its interaction with the OS, are crucial to achieve high performance. From the point of view of an SFI emulator and EFU that are based on commercially available computer systems, we can conclude that the above described issues become indispensable requirements.

6.5.2 SFI Emulator

The performance of the fragment layer of this SFI emulator, regardless of its simple implementation, is quite satisfactory. It can be improved further if the number of fragment copying can be reduced. The obvious way to do that could be to eliminate the Fragment Buffer as an intermediate fragment storage. To achieve that, some sort of fragment processing will be required to be done in the NIC. Only few commercially available ATM NIC offer today such possibilities. It is believed however, that they will become more available in the future, as the on-board protocol processing may alleviate the specific to ATM overheads due to the multiple network layers. It is also not clear, if for instance IP protocol processing can be useful to event assembly.

Another way to increase the event assembly performance is to increase concurrency. The possibility to have more than one execution thread in the fragment management layer, was also considered in the initial design phase of this emulator. However, for simplicity reasons and also to have a clearer view of the worst case performance, it was not implemented. One possibility is the fragments of different events to be sent to the ATM VC equal to the event number modulo the highest supported VC. In addition, the device driver can be modified to store incoming data in different buffers for each VC. That way, several event-assembly threads can be started, each one accessing different incoming fragment and event data buffers and thus, minimizing also the synchronization overhead. The obvious disadvantage of that possibility is that it heavily relies on the ATM protocol internals.

From the ATM measurements we performed, it was clearly shown that utilizing larger AAL5 packets, the effective bandwidth can be increased. The choice of the AAL5 packets with size equal to the event fragment size plus a short header, avoids the additional layer of an event building protocol and also the possibility of having multiple fragments in one AAL5 packet. A non-reliable version of such protocol may not perform well, because a single cell loss will result to several lost events. A reliable event building protocol *e.g.*, TCP/IP, could be one choice. However, the experience with TCP/IP over ATM so far [Fouquet], has not been encouraging enough to make such a solution attractive. Also, TCP/IP implies retransmissions and acknowledgments, something that is not currently considered in the design of the RDPM and SFI.

6.6 Summary

We have considered an EFU prototype setup, based on a commercial SMP computer system, connected to an ATM network. A software emulator of the SFI has been developed and tested in this setup. The performance issues of the ATM network, NIC adapter and the SFI emulator have been extensively studied. It has been shown that a simple implementation of a software SFI emulator, can easily meet and even exceed the currently required performance. It was shown that the throughput achieved by this emulator was limited by the effective performance of a high speed network and by the available memory bandwidth, as seen from a user application.

It is appropriate to suggest that an emulated SFI can meet the performance requirements settled by the CMS DAQ architecture. A hardware based SFI may provide better performance, than the emulated SFI. However in addition to the higher development and maintenance costs, a hardware SFI still will depend on the network performance.The I/O performance turns out to be a major problem that will also hinder the performance of a custom SFI built in hardware. In order to achieve the I/O performance required by the EFU, new and simpler host interfacing to the NIC is required.

7 Farm Scaling

For an optimum design of the CMS DAQ system many configurations of the event builder and the event filter farm will have to be studied and evaluated, either by simulations or prototypes or both. The aim of this optimization is to find the best trade-off between construction cost and system performance, taking into account the actual or extrapolated technological evolution. An interesting aspect of this optimization problem is the matching of the number of event filter units (EFU) and consequently the event builder (EVB) switch ports, with the number of processors in each of the EFU.

In this chapter the scaling considerations of the EVB and event filter farm (EFF) are laid out, together with the resulting consequences taking into account today's extrapolations of the computer and communications technology evolution.

7.1 Scaling Parameters

Building the optimum system of the EVB and EFF, a number of configuration parameters can be assumed to vary, when specific sets of system requirements, technological solutions and available funds have to be matched. A major obstacle in this optimization is the little knowledge on a targeted technology for the EVB and the EFU. However, certain performance properties of those systems can be exposed when some of those configuration parameters are varied.

Two approaches to build the EVB and EFF can be distinguished. With the *large-farm* approach, relatively many switch ports and EFU can be used with each of the EFU containing a small number of processors (one to three). Conversely, with the *small-farm* approach, less switch ports and EFU can be used with more processors in each of the EFU. Both approaches must be based on the assumption that the necessary EFU and switch technologies are available to in order to accommodate the increased bandwidth, event rate and processing power requirements. An interesting question to answer is what are the relative merits of those two approaches. An answer to this can be given by examining the consequences of these two approaches on the individual requirements of the EVB and the EFF.

7.1.1 Event Builder Size

The EVB switch has to interconnect an unusually high number of input and output ports and offer a total bandwidth of several hundreds of gigabits per second. These two requirements of the EVB switch are also addressed by projects in the gigabit networking research world [Part94] but also fit well into what the modern networking research is aiming at [Part94a]. The high number of switch ports issue is typical to a series of recent research projects that have successfully built networks of workstations (NOW) of up to 100 nodes with close to supercomputer performance [Ande95] [Chio96] [Mark96]. The appearance of fast switched network architectures like ATM, Myrinet [Myri95] and Fast Ethernet, have made the above possible. The quest of higher bandwidth switches receives significant attention by many research proposals an prototypes. Amongst them, projects like TORUS [Gend97] and S-Connect [Nowa95] [Nowa95a] are addressing a Terabit per second ATM switch and optical crossbar interconnect respectively.

According to [Chan96] and [Fing96] large size, high bandwidth switches with very good internal blocking behavior, are becoming feasible to build. Large switches are very interesting choices for big local or campus area networks, hence it is possible that commercially available and yet affordable products appear in the near future.

The current understanding of the event building utilizing large switches, suggests that an equal number N_i of input and N_o output ports might perform best, assuming some input traffic shaping. Higher link bandwidth could result into switches of smaller size *i.e.*, less number of ports. It is interesting to know what determines the lower limit of number of switch ports when $N_i = N_o$.

The N_i, has a lower limit posed by the capabilities of the RDPM. The RDPM internal data-transfer speed must be higher than their input and output link speeds. The internal speed consequently is limited by the actual implementation of the RDPM (currently based on PCI). The N_o is expected to be less limited than the N_i, because it does not depend on any built-in limitations, like in the case of RDPM. N_o is actually determined by the available networking capability of computer systems. As the need for higher networking speeds increase, it seems reasonable to assume that I/O speeds at the gigabit range in computer systems will become available.

Given a lower value of N_i, one could consider the case where N_o can be smaller than N_i. This case although cannot be excluded, might have serious implications in the event building latency. Those implications have yet to be understood for each of the communication technologies considered for the switch and the various traffic shaping mechanisms.

Technically, it seems possible that smaller square switch configurations will become feasible with the advances of the networking industry. What remains to be seen is the effect on the cost choosing a higher bandwidth and smaller size switch, instead of lower bandwidth and larger switch. In [Witt92] an attempt is done to characterize the per-port scaling of the switch cost for various ATM network architectures, based on an assumed model for the chip package count. It is shown that there is a significant disparity across the studied switch architectures and sizes, with the most competitive architectures keeping almost constant the per port chip count for both small and large sizes.

7.1.2 Number of EFU and Processors

From the performance point of view, a small and a large EFF, with the same total number of processors, are not expected to result into systems with different processing capacities. We have performed simulations of such systems using the simulator described in Chapter 5, confirming that. The two systems however, pose different requirements on the individual EFU.

As an illustration, let us consider the case of a total 3,000 processors required in the EFF. We can build then a large EFF system of 1,000 EFU and 3 CPU each and a small EFF system with 500 EFU and 6 CPU each, or even 375 EFU with 8 CPU each. Obviously, in the case of smaller EFF configurations, it is required that SMP systems scale well in the three to ten CPU range. Even in the conservative case of a triggering software that benefits very little from a parallel architecture, a significant scaling of the memory bandwidth will be needed, just because of the increased I/O bandwidth.

If we take into account widely available commercial systems and eventually commodity systems, it is important to see up to how many processors such systems are typically marketed. Most of the today’s mid-range systems have two to four CPU. The cutting edge between mid-range and high-range systems seems to be around six CPU per system, where larger configurations can accommodate up to 36 or even 64 CPU. In other words, even in the case of a linear system scaling in the three to ten CPU range, the cost will not scale linearly, but instead it will grow in steps, determined by the current marketing trends.

It is therefore essential to know such limits (which eventually may grow in the next five years) because adding more CPU per EFU may result into the purchase of more expensive systems than actually needed.

An eventual variation of the small-large approaches is the middle choice, to have more than one switch ports connected to each of the EFU. This possibility can be considered in the case that the EVB and EFF are scalable but the input part of the EVB *i.e.,* the RDPM cannot scale. Obviously, the demands on the memory and I/O bandwidth will be higher in that case. Such a possibility however, could make smoother the scaling to larger systems as was discussed above.

From the reliability point of view, it is more economic to increase the fault tolerance by using less EFU (*e.g.,* an additional power supply to each node). An additional consideration here is the impact of occurring faults. On one hand, less EFU mean lower failure probability and more EFU will result in increased number of failures. On the other hand, failures of EFU with more CPU will have a much higher impact as the number of events queued for or being under processing will be larger. The failure impact will clearly be much higher in the case of EFU connected to more than one switch ports, rendering more than one destinations unusable at the same time.

7.1.3 I/O Bandwidth

The most obvious difference between a small and a large EFF system, is the I/O bandwidth needed by the SFI and the EFU. In the small EFF system, the input data rate per EFU will be higher than in the large system. This is probably the most important performance consideration when trying to understand which system is better.

We have seen in Chapter 6 that getting the I/O bandwidth to the LV2 and LV3 trigger software, is a difficult task mostly due to the presence of the OS limitations. Therefore, in the case of a small EFF system, where larger EFU systems are used, more effort will be required to guarantee that the required I/O bandwidth can be delivered.

The same I/O performance problems are expected to appear also when custom designed SFI are used. The SFI internal bandwidth is also limited, the same way as it is in the RDPM. Therefore, an EFF system built with custom designed SFI, is expected to be less flexible to operate in a large EFF configuration.

7.1.4 Processor Performance

Throughout this work we have assumed the simplification of including the CPU performance into the service time of a LV2 and LV3 trigger process. This simplification was mostly justified because of the very limited current knowledge of the characteristics of the trigger software.

There is another reason why the detailed analysis of the trigger algorithms might not lead to realistic results. The triggering software will be developed not only prior to the start of the experiment, but also during its operation. Newer versions of the triggering algorithms, may add to the workload of the EFU. Therefore, an uncertainty factor of the needed processing capacity is needed to be taken into account.

The average LV1-trigger rate is another parameter determining the needed CPU capacity. So far, it was considered for it only the 100 kHz design maximum value. The actual LV1-trigger rate is expected to vary, from lower values at the initial running period of the experiment, to higher values when the LHC luminosity reaches its nominal value. At the lower luminosity operation phase of LHC, the processing requirements on the EFF will be lower. A later upgrade of the EFF, to accommodate the needs of the high luminosity period, may result in a heterogeneous farm if different EFU are added. The architecture of the EFF permits such heterogeneous configurations, although they were not considered in the simulations. This is a convenient and economic way to scale the EFF as new needs arise.

Regarding the farm scaling as the processor speed is increased, the same performance considerations apply for uniprocessor (UP) and multiprocessor (MP) systems. That is, an increase of the processor speed makes both MP and UP systems with better performance, while making bigger MP systems does not improve the UP performance. Apparently there is no evidence that the ultimate processor speed one day will be reached, it is only the different technologies that are limited. The CPU speed increase is a continuous challenge.

Estimates of the processor performance growth, show an 80% yearly increase [Bask91]. The memory speeds are estimated to have a much lower increase rate of 7% [Henn90] every year. From those two performance evolution figures, the question of well balanced systems arises. As described in [Wulf95] such a discrepancy of the increase rates will lead into average cache access times equal to the main memory access cost, which consequently might result into no performance improvement when the processor speed is increased. Such a problem could appear by the starting date of the CMS experiment, if the technology does not change. Those arguments however, are probably pointing to that the edge of the current technology has been reached and an indication of an eminent leap in computer evolution.

7.2 Scaling Scenarios

Varying the number of EFU in the EFF, it will result in farm configurations with different requirements on the EFU I/O bandwidth, but also in different requirements on the network speed of the EVB switch. To evaluate the I/O requirements of the EFU when the EFF is changed from smaller to larger configurations, we need also to take into account the EVB switch bandwidth.

The network speeds offered by the considered technologies (ATM, FC) are in the 100 Mb/s to 10 Gb/s range. In both ATM and FC technologies the increase of speed is done in steps, for instance in ATM, OC-3, OC-12, OC-48, etc. link speeds are today possible. Changing from OC-12 to OC-48 apart from the additional capacity it will offer, it will also have significant financial consequences.

7.2.1 LV1-trigger Maximum Rate

We can obtain a figure of the maximum LV1-trigger rate λ_{LV1}^{max} that a particular farm configuration can handle, if we take into account the sustained bandwidth of the EFU input as given by (F 7.1), which essentially represents the sum of the bandwidth of each of the LV2 and LV3 data streams to the EFU.

$$B \geq \frac{\lambda_2 S_2 + \lambda_3 S_3}{k} \qquad \textbf{(F 7.1)}$$

With the aid of (F 7.1) and taking into account (F 7.2) how the LV2 rate λ_2 relates to the LV1 rate λ_1 and the rejection factor *R*,

$$\lambda_2 = \frac{\lambda_1}{R} \qquad \textbf{(F 7.2)}$$

as well as the relation of the total event size *S* with the LV2 event size S_2 and LV3 event size S_3 and

$$S_3 = S - S_2 \qquad \textbf{(F 7.3)}$$

the relative size *p* of the LV3 event part S_3 given by (F 7.4),

$$p = \frac{S_3}{S} \qquad \textbf{(F 7.4)}$$

we arrive to a relation (F 7.5) for the maximum sustainable LV1 trigger rate.

$$\lambda_{LV1}^{max} \leq \frac{kB}{S} \cdot \frac{1}{\frac{p}{R} - p + 1} \qquad \textbf{(F 7.5)}$$

It should be noted that (F 7.5) does not take into account any processing time for LV2 or LV3. It simply shows an upper limit to the maximum LV1 trigger rate set by system limitations.

Using (F 7.5) we will study several cases of communication technologies and identify where the limits fall.

Two values of *R* are considered, 10 and 20. Although they are less than what is expected to be the real case, they give a worst case result for the maximum LV1-trigger rate. The ratio *p* takes three values, the current assumption of 75% of the total event, one lower (65%) and one higher (85%).

In the following figures, the horizontal axis represents the bandwidth needed to sustain both LV2 and LV3 event-part streams at the SFI input (the EVB switch output port). They gray area of each figure represents the limits set by the 100 kHz LV1-trigger rate and the maximum effective bandwidth that can be delivered by a given network link speed. An ATM network is assumed, that can operate at OC-12 (622 Mb/s) or OC-48 (2.4 Gb/s) link speeds.

The effects of traffic shaping on the resulting output data-stream are not taken into consideration. No protocol above the ATM adaptation layer is considered and consequently a non-congested EVB switch is assumed.

In Figure 7.1 the case of a switch with a 1,000 ports is shown. Each port runs at an OC-12 link speed, resulting in a 430 Mb/s maximum effective bandwidth. It is shown that for all considered values of *R* and *p*, the needed I/O bandwidth at the EFU is well below the maximum effective bandwidth for the maximum LV1-trigger rate.

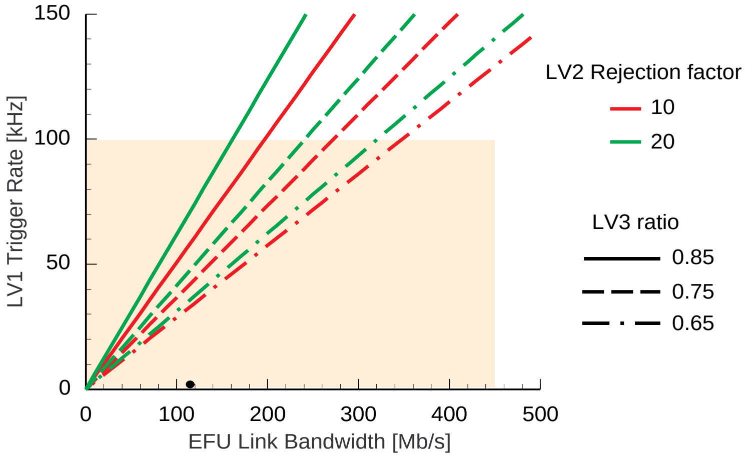

Figure 7.1: 1000 switch ports at OC-12 speed

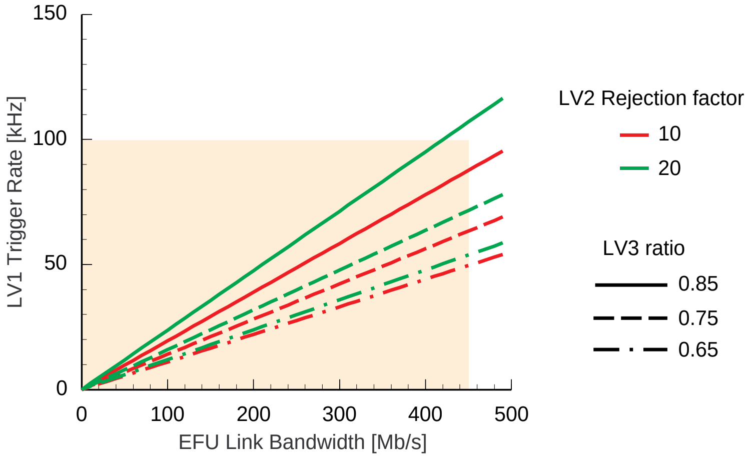

Figure 7.2: 500 switch ports at OC-12 speed

In Figure 7.2 and Figure 7.3, a 500 port and a 384 port switch are shown, respectively. In the 500 port case, only when *p* is 0.85 the maximum LV1-trigger rate can be sustained, although marginally. In the case of 384 ports, *p* has to be equal to or more than 0.85 and the rejection factor *R* equal to or more than 20. Both cases are marginal if a 100 kHz LV1-trigger rate has to be sustained.

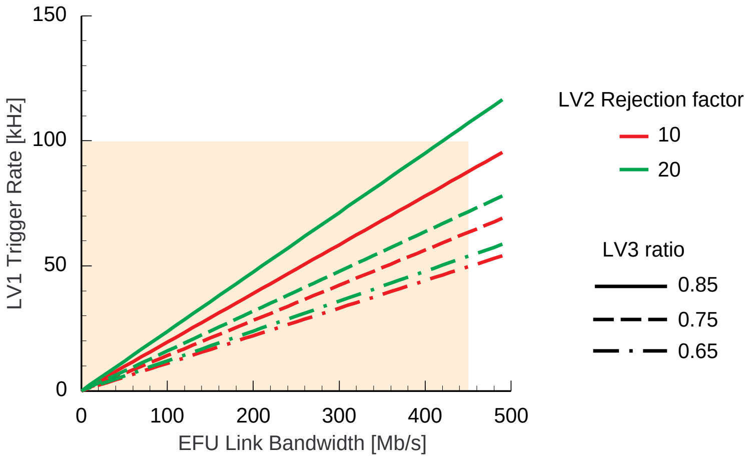

Figure 7.3: 384 switch ports at OC-12 speed

Similarly, Figure 7.4 shows the case of a 256 port EVB switch, this time using an OC-48 link speed. All combinations of *R* and *p* are well below the maximum effective link bandwidth, which is approximately 1.8 Gb/s.

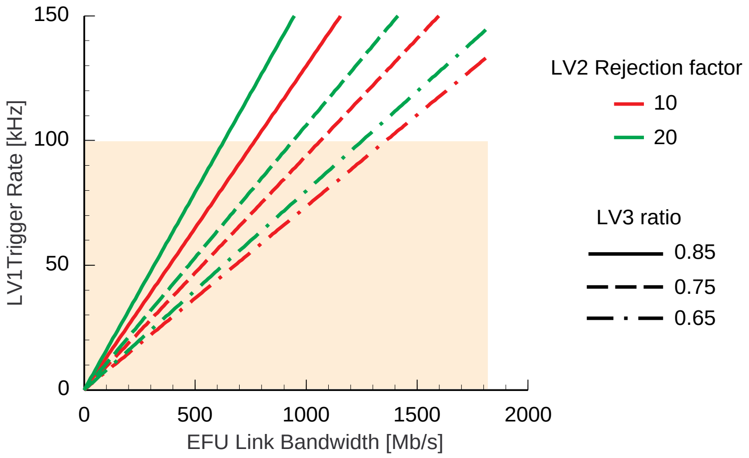

Figure 7.4: 256 switch ports at OC-48 speed

Those results are independent of the LV2 and LV3 trigger service times, as they simply reflect the effect of the individual LV2 and LV3 rates. Clearly there is a limit of the OC-12 technology to build systems of around 500 ports. Smaller configurations of the EVB and EFF will require an OC-48 technology to be used, if the maximum LV1-trigger rate of 100 kHz has to be respected. Otherwise, the maximum sustained LV1-trigger rate could be reduced to a values lower than 100 kHz, which might result to lower physics performance of the detector.

7.2.2 Effect of the LV2 Rejection Factor

Apart of the required CPU power to run a particular set of the LV2 and LV3 trigger algorithms, there is the LV2-trigger rejection factor *R* that also determines the load of the EFF. The LV2 rejection factor determines the frequency of incurring LV3 jobs, which are expected to have service times of several times more than that of the LV2 jobs.

It is useful to know what is the effect in the EFF when *R* is varied. As it was shown in the previous paragraph, EFF configurations with approximately 500 nodes, may not be able to sustain the maximum LV1 trigger rate. Therefore other parameters as the trigger service time and rates may needed to be reduced.

The bandwidth required at the SFI/EFU input can be modeled as a function of the LV2 rejection factor *R* as shown in the relation (F 7.6) which is obtained by (F 7.5).

$$B_{EFU} \geq \frac{\lambda_{LV1} \cdot S}{k} \cdot \left(\frac{p}{R} - p + 1\right) \qquad \textbf{(F 7.6)}$$

Similar to (F 7.5) we denote with λ_{LV1} the LV1-trigger arrival rate, *S* the full event size, *p* the ratio of the LV3-part event size over the full event size *S* and *k* is the number of EFU nodes in the farm.

The dependency of the B_{EFU} from *R*, is inversely proportional. In Figure 7.5, we have plotted few values of *k* (384, 500, 625 and 1,000 nodes) when *p* is equal to 0.75 and λ_{LV1} is equal to 100 kHz.

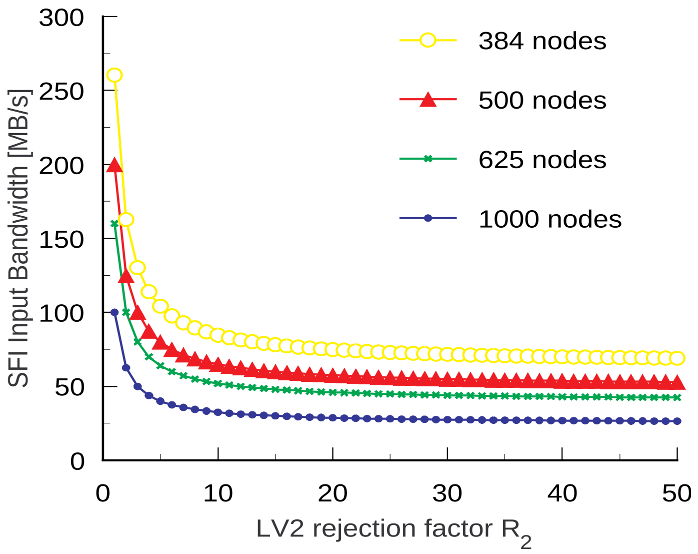

Figure 7.5: The SFI input bandwidth dependence from the rejection factor

We can see that with that set of values, the range of the rejection factor values that we are interested on (from 20 to 100) are lying on the asymptotic part of the plots. If we increase the rejection factor from 50 to 100, the SFI/EFU input bandwidth will be reduced in average by one MB/s only. It will be reduced by another MB/s if we additionally increase *R* from 100 to 150.

The EFU/SFI input bandwidth is much more variable, when *R* takes values less than 10. This range of R however, is excluded as the LV2 trigger is expected to have higher rejection rates.

The dependency of the SFI/EFU input bandwidth from the rejection factor of the LV2 trigger algorithms, is actually indicating that in situations as that shown in Figure 7.2, very little can be gained in terms of resulting SFI/EFU input bandwidth, if *R* is increased. Therefore, at either the LV1 trigger rate has to become lower than 100 kHz or faster network has to be deployed.

7.3 Conclusion

In this chapter we tried to illustrate the problems related to the performance of the EVB and EFF when their sizes are varied. The case of a small and a large farm were considered as two possible approaches to build the EFF of the CMS DAQ system. From the performance point of view, given that the necessary communication technology is available, the farm performance can meet the requirements. It was shown that for the currently considered LV1-trigger rate and LV2-trigger rejection factors, smaller sizes of the EFF will require the use of at least OC-12 ATM network technology. A quantitative evaluation of the LV2-trigger rejection factor effects on the resulting network bandwidth, was done. It was shown that the required switch output bandwidth is very little affected when the rejection factor of the LV2-trigger is varied, in the range of values (> 20) considered in the current EFF design.

The cost factor in the small and large EFF configurations, depends mostly on the switch ports and the number of EFU. It seems appropriate to suggest that a choice between a small and a large farm will have to be based more on the per EFU costs and less on the switch per port costs.

8 Conclusions and Prospects

The aim of this work was to expose the aspects of constructing a large scale computer farm based on MP systems, for the on-line data processing purposes of the CMS experiment. Two approaches were followed. One using a discrete-event simulation model of the farm and a second using a prototype environment for the EFU. Both approaches together, were used to set a framework for the performance evaluation of different design options of the farm. The conclusions from the presented work influence not only the overall DAQ architecture of CMS, but also that of the event filter unit (EFU).

8.1 Impact on the CMS DAQ Architecture

From the simulations shown in Chapter 5, it was shown that the modular design of the event filter farm (EFF) can easily adapt to different assumptions for the EFU performance and respectively to the LV2 and LV3 filter processing loads. The simulation model was assuming an aggressive way of requesting new events to be processed by the EFU. That way, the read-out units (RU) can be easily off-loaded, given that enough buffering of events is available in the EFU. This method can offer the possibility of a feedback to the LV2 or LV3 trigger algorithms depending on the current size of the LV2q and LV3q, which consequently can modify their rejection factor.

The simulation model that was described, assumes a simple representation of the network connection between the EFU and the event manager (EVM). A relatively high throughput of messages sent by all the EFU must be able to be sustained at the EVM end-point and by the transport network itself.

The current DAQ architecture assumes a separate device for the switch–to–farm interface (SFI). The prototype studies shown in Chapter 6, give a good evidence that its functionality can be emulated by software in each of the EFU. It was shown that the maximum throughput achieved by the emulator is well above the required 100 events per second, in the worst case of input parameters. When more realistic input parameters are used, the event assembly throughput is increased sufficiently to accommodate smaller farm configurations consisting of 512 EFU.

The event assembly throughput of the presented SFI emulator, was essentially limited by two factors. These are the number of event fragment copies that are required by the emulator implementation and the available memory bandwidth of the EFU.

Taking into account the evolution of computer systems performance, it is safe to assume that the memory bandwidth of future computer systems is going to improve significantly. This is because main memory access times are already today lagging behind the processor speed and any further improvement of the last will have very little total effect if the memory bandwidth stays the same.

In Chapter 5 were described also alternative ways to reduce the number of data copies required for the assembly of a LV2 or a LV3 sub-event. This is an obvious optimization that requires a better design of the network layer of the SFI emulator, according to the available capabilities of the utilized network interface controller (NIC).

The results of the SFI emulation, indicate that a significant simplification of the DAQ architecture could be done. The construction of a separate device for the SFI can be avoided. Such a simplification will result not only into reduced construction costs, but also into minimum maintenance during the operation of the DAQ system. It will also increase the portability, as there will be almost no dependence on the EFU hardware architecture. The flexibility offered by an SFI emulator running into the EFU, must also be accounted. Optimizations of the event building system and particularly of the event assembly into sub-events, can be done much easier in the emulator than in a hardware device. The SFI emulator also does not depend on a particular network technology, hence it enables prototypes of the DAQ system to be built with a little effort.

In Chapter 7, were outlined the different factors influencing the EFF size. Two possible ways to build the EFF were identified. A small farm using less but powerful EFU and a large farm with many but less powerful EFU. It was shown that in order to satisfy the requirement that the EFF can handle an input rate of 100 kHz, smaller configurations will require faster networking technology to be available. In particular, EFF configurations smaller than what is considered today (1000 EFU), will be at the limits of the available speed of the OC-12 ATM networks. A quantitative evaluation of the resulting bandwidth at the inputs of the EFU was done, when the LV2-trigger rejection factor is varied. It was shown that there is very little reduction of the required bandwidth when the rejection factor is increased more than 10.

The economics of the EFF scaling can be expressed by the effects of the EVB switch technology and per port costs and by the EFU cost itself. The cost associated with the switch technology may become large if the speed requirements are at the limits of a lower speed technology and a faster communication network has to be used. The switch per port costs can be as low as constant when its size is varied. Finally the EFU costs, are increasing as the EFF becomes smaller, with the costs of more sophisticated network adapters adding to it.

8.2 Impact on the EFU

The EFU that was considered throughout this work, was an SMP computer system. From its architecture point of view, an SMP–based EFU was represented as a system that its throughput of finishing LV2 and LV3 jobs increases when more processors are added to it. Also, the job scheduling can take place with equal probability on any of its processors.

Two issues were distinguished as being of paramount importance for the EFU. Firstly, it was the way that the LV2 and LV3 filtering jobs are scheduled, and secondly, the EFU interfacing with the event builder (EVB) network.

In Chapter 5, a simple model of an SMP–based EFU was used, to study the EFU behavior. The LV2 and LV3 filter workloads were modeled as a flow of arriving jobs at each EFU. The handshaking that was used between the EFU and the RU, was aggressively requesting new LV2 jobs in order to off-load as much as possible the RU. With the assumptions for the LV3 processing time to be on average 100 times the LV2 processing time, and the rejection factor of the LV2 trigger to be a factor of 30, the preemption of LV3 jobs by the LV2 jobs turns to be a sensible choice for the EFU scheduling. The LV3 preemption can increase the service rate of LV2 jobs, by increasing the response time of the LV3 jobs. A response time longer than the service time for the LV3 jobs can be tolerated, given sufficient memory resources in the EFU. However, under heavier load of LV3 jobs, there must be a way to limit how many times a LV3 job can be preempted. This stability condition was identified analytically but it was not accounted by the model used for the EFU. The EFU model can be trivially modified to take account of that.

With the prototype setup described in Chapter 6, issues arising from the use of modern SMP computer systems as an EFU, were studied. In particular, with the aid of the SFI emulator, the issues of interfacing a user application with a network interface controller (NIC), became apparent. For the purposes of the SFI emulator, the traditional way of interfacing an NIC through the operating system was used. This turned to be inappropriate, as it could not satisfy the needs of bandwidth and data transfer latency required for the fragment assembly into events. The throughput of data transfers that are expected to take place in an EFU with SFI emulation, is particularly important for small data packets. It is important to note that similar performance requirements are also needed in the case of a custom-made SFI device connected to the EFU.

Therefore it seems that a way of receiving data packets from the NIC directly into the user application data buffers, can offer much better performance. From the architecture point of view, the receiving latency can be improved if the NIC can be attached closer to the processor. There are many proposals in the direction of making the NIC less peripheral, like making it cache-coherent and attaching it on the memory bus.

Modern research in high speed network interface architectures has identified the latency and bandwidth performance issues as crucial performance parameters and significant improvements of future systems can be expected. Such architecture improvements are an absolute necessity for the operation of the EFF foreseen in the CMS experiment.

8.3 Future Work

This work has been started at the very early design phase of the CMS experiment's data acquisition system. The evolution of the CMS DAQ design during the coming years and until it is finalized, requires more studies to be done for the EFF and the EFU in many directions. In particular the technology evolution in the areas of communication and computing systems must be closely followed.

It seems appropriate that a detailed simulation of the EVB switch for a variety of communication technologies and in the simulation framework of Chapter 5, is necessary in order to prove that a large scale switch can operate successfully. In particular the issues of output blocking, bandwidth efficiency and the switch fabric scalability, require deeper understanding for each of the considered communication technologies. Results from such studies will also provide additional information on the profile of the input data streams to the EFF.

The problems appearing in the high performance I/O interfacing of computer systems, require an evaluation methodology to be developed, as the demands for higher communication speeds increase, the computer hardware evolves and the operating systems offer new I/O interface facilities. The I/O issue receives continuously more attention among researchers, with many new ideas developing.

The evaluation of MP systems for the EFU, will require also more precise information on the characteristics of the filtering algorithms. This will enable studies in both directions of evaluating systems and improving the algorithms themselves.

List of Figures

Figure 1.1: CERN accelerator complex 16
Figure 1.2: Cross-section of the LHC dipole 17
Figure 1.3: Perspective view of the CMS detector 18
Figure 1.4: Architecture of the CMS DAQ system. 22
Figure 2.1: Event building using a switch network 26
Figure 2.2: RDPM functional diagram 27
Figure 2.3: Barrel shift network 30
Figure 2.4: Clos network with 3 stages. 32
Figure 3.1: Multiprocessor classification 42
Figure 3.2: SMP Architecture 43
Figure 3.3: Virtual Memory layout 46
Figure 3.4: Memory Hierarchy 48
Figure 3.5: Bus based SMP system 52
Figure 3.6: Cross-bar based SMP system 53
Figure 5.1: Functional decomposition of the DAQ system 70
Figure 5.2: Overview of the DAQ simulation model 71
Figure 5.3: SFI and EFU model 72
Figure 5.4: Simulated switch LV2 latency 77
Figure 5.5: Simulated switch LV3 latency 77
Figure 5.6: Pointer Buffer Table Usage 81
Figure 5.7: Farm Request (FRQ) length 82
Figure 5.8: Arrival rate of LV2-read and LV3-read requests 83
Figure 5.9: SFI buffer occupancy 84
Figure 5.10: LV2q queue length 84
Figure 5.11: LV3q queue length 85
Figure 5.12: LV2 decision latency 85
Figure 5.13: LV3 decision latency 86
Figure 5.14: Number of interrupts per LV3 job 87
Figure 5.15: Farm CPU utilization 87
Figure 5.16: Farm instantaneous output rate. 88

Figure 6.1: Model of the SFI emulator 90
Figure 6.2: Prototype setup 93
Figure 6.3: Architecture of the SFI emulator 95
Figure 6.4: AAL5 packet layout 97
Figure 6.5: AAL5 segmentation into ATM cells 97
Figure 6.6: ATM AAL5 Theoretical and measured performance 98
Figure 6.7: Packet receive latency 100
Figure 6.8: Memory bandwidth and scaling 101
Figure 6.9: Event Assembly Performance 103
Figure 6.10: Profile of the main thread 105
Figure 6.11: Profile of the event building thread 106
Figure 6.12: Profile of the event consumer thread 107
Figure 7.1: 1000 switch ports at OC-12 speed 119
Figure 7.2: 500 switch ports at OC-12 speed 119
Figure 7.3: 384 switch ports at OC-12 speed 120
Figure 7.4: 256 switch ports at OC-48 speed 120
Figure 7.5: The SFI input bandwidth dependence from the rejection factor . . . 122

List of Tables

Table 1.1: Fundamental forces and their characteristics 14
Table 1.2: Fundamental elementary particles 15
Table 1.3: CMS detector channels and event sizes 19
Table 1.4: LV1 trigger rate break-down 21
Table 1.5: Summary of the CMS DAQ parameters 25
Table 2.1: RDPM main characteristics 29
Table 3.1: MP Systems 43
Table 3.2: Processor architecture evolution 45
Table 5.1: Fixed simulation parameters 74
Table 5.2: RDPM parameters 76
Table 5.3: Variable simulation input parameters 80
Table 6.1: Performance of the workstations used in the setup 93
Table 6.2: Structure of the emulated stream of LV2 and LV3 fragments 94
Table 6.3: Performance of the emulator in DA mode 104
Table 6.4: NIC Attachment points in commercial computer systems 109

List of Acronyms

A

ATM	Asynchronous Transfer Mode	23

C

CERN	European Center for Particle Physics	15
CMS	Compact Muon Solenoid	17

D

DA	Deferred Assembly	102

E

EFF	Event Filter Farm	22, 35
EFU	Event Filter Unit	22, 37
EVM	Event Manager	22, 34

F

FC	Fibre Channel	23
FED	Front-End Driver	22
FPq	RDPM Free Pointer Queue	28
FRB	RDPM Filter Request Buffer	28
FRq	RDPM Farme Request Queue	28
FRS	Filter Resource	34

G

GTS	Global Trigger System	20

I

IEq	RDPM Input Event Queue	28

L

LHC	Large Hadron Collider	16
LV1	Level-1 Trigger	20
LV2	Level 2 Trigger	21
LV3	Level 3 Trigger	21

O

OED	RDPM Output Event Descriptor	28

P

PBT	RDPM Pointer Buffer Table	28

R

RDPM	Read-out Dual Port Memory	22
RU	Read-out Unit	22

S

SA	Simultaneous Assembly	102
SCI	Scalable Coherent Interface	23
SeqIN	RDPM Input Sequencer	28
SeqOUT	RDPM Output Sequencer	28
SFI	Switch-to-Farm Interface	33
SM	Standard Model	14

T

TS	RDPM Task Scheduler	28
TTC	Trigger, Timing and Control	20

References

Chapter 1

[ATLA94] ATLAS collaboration, *ATLAS Technical Proposal*, CERN/LHCC 94-43, 1994

[CDF96] CDF Collaboration, *The CDF II Detector: Technical Design Report*, FERMILAB PUB-96/390-E, 1996

[Citt95] Cittolin S., Gillot J.F., Ràcz A., Halsall R., Haynes B., *Front End Driver in CMS DAQ*, CMS TN/95-020, Version 1.03, February 1995

[CMS94] CMS collaboration, *CMS Technical Proposal*, CERN/LHCC 94-38, 1994

[CMS96] CMS Collaboration, *Technical Proposal for CMS Computing*, CERN/LHCC 96-45, 1996

[LHC95] *LHC Conceptual Design*, CERN/AC 95-05, 1995

[Lack95] Lackey J., et al., *CMS Calorimeter Level 1 Trigger Conceptual Design*, CMS TN/94-284, January 1995

[Neum97] Neumeister N., et al., *CMS Global Trigger, Preliminary Specifications of the Baseline Trigger Algorithms*, CMS Note 1997/009, January 1997

[Tayl95] Taylor B. G., *TTC Distribution*, In Proceedings of the 1st Workshop on Electronics for LHC Experiments, CERN/LHCC/95-56, pp. 180-184, Lisbon, 11-15 September 1995

Chapter 2

[Ahma89] Ahmadi H., Denzel W., *A Survey of High-Performance Switching Techniques*, IEEE Journal on Selected Areas in Communication, Vol. 7, No. 7, September 1989

[Bars90] Barsotti E., Booth A., Bowden M., *Effects of Various Event Building Techniques on Data Acquisition System Architectures*, Invited talk presented at CHEP'90, Santa Fe, New Mexico, April 9-13, 1990

[Bran95] Branson J., Fisk I., Mojaver M., *Development of a C80 Multiprocessor Based DPM: VORTEX*, Presented at the LHC DAQ Workshop, CERN, Geneva Switzerland, March 29-30, 1995

[Baue96] Bauer G., et al., *CDF DAQ Upgrade and CMS DAQ R&D: Event Builder Tests Using an ATM Switch*, Presented at 2nd International Data Acquisition Workshop on Networked Data Acquisition Systems (DAQ 96), Osaka, Japan, November 13-15, 1996

[Bian93] Bianchi G., Turner J., *Improved Queueing Analysis of Shared Buffer Switching Networks,* IEEE ACM Transactions on Networking, August 1993

[Chan96] Chaney T., Fingerhut J. A., Flucke M., Turner J. S., *Design of a Gigabit ATM Switch*, WUCS-96-07, Washington University, St. Louis, USA, 1996

[Citt95] Cittolin S., Fucci A., Sphicas P., Sumorok K., *Dual Port Memories in LHC Experiments,* CMS-RD12 TN-95-04, Version 1.00, 1995

[Fahm95] Fahmi S., *A survey of ATM Switching Techniques,* Department of Computer and Information Science, Ohio State University, August 1995

[Fucc95] Fucci A., Gigi D., *Hardware Implementations of Dual Port Memories for CMS Experiment*, CMS-RD12 TN 95-05, 1995

[Gell95] Gellrich A., et al., *The Processor Farm for Online Triggering and Full Event Recostruction of the HERA-B Experiment*, Poster at CHEP'95, Rio de Janeiro, Brasil, 1995

[Goke73] Goke L. R., Lipovski G. J., Banyan networks for partitioning multiprocessor systems, in Proc. 1st Annu. International Symposium on Computer Architecture, p. 21 -28, December 1973

[Pryc94] de Prycker J. M., *Asynchronous Transfer Mode*, 2nd ed., Ellis Horwood Series in Computers and their Applications, 1993

[Turn97] Turner J., Yamanaka N., *Architectural Choices in Large Scale ATM Switches*, WUCS-97-21, Washington University, St. Louis, May 1997

Chapter 3

[Dijk68] Dijkstra E. W., *Co-operating sequential processes, In Programming languages*, F. Genuys (ed.), p. 43 - 112, Academic Press N.Y., 1968

[Flyn66] Flynn M. J., *Very high speed computing systems*, Proceedings of the IEEE, vol. 4, p. 1901 - 1909, 1966

[Henn90] Hennesy J. L., Patterson D. A., *Computer Architecture a Quantitative Approach*, Morgan Kaufmann Publishers Inc., San Mateo CA., USA, 1990

[Henn91] Hennesy J. L., Jouppi N. P., *Computer Technology and Architecture: An Evolving Interaction*, IEEE Computer, 24, (9), p. 18-29, 1991

[Kell91] Kelly E. G., *SPARC MBus Interface Specification*, Sun Microsystems Inc., March 1991, Revision 1.2

[Mash82] Mashburn, H. H., *The C.mmp/Hydra Project: an Architectural Overview*, Computer Structures: Principles and Examples, by D.P Siewiorek, C. G. Bell, A. Newell, Ch. 22, McGraw-Hill NY, USA, 1982

Chapter 4

[Fish95] Fishwick P. A., *Simulation Model Design and Execution: Building Digital Worlds*, Prentice-Hall, ISBN 0-13-098609-7, 1995

[Jain91] Jain R., *The Art of Computer Systems Performance Analysis*, J. Wiley & Sons Inc., ISBN 0-471-50336-3, 1991

[Klei75] Kleinrock L., *Queueing Systems; Volume I: Theory*, John Wiley & Sons Inc., ISBN 0-471-491101, 1975

[Klei76] Kleinrock L., *Queueing Systems; Volume II: Computer Applications*, John Wiley & Sons Inc., ISBN 0-471-49111-X, 1976

[Mars88] Marsan M. A., Balbo G., Conte G., *Performance Models of Multiprocessor Systems*, 2nd printing, MIT Press, ISBN 0-262-01093-3, 1988

[McCa95] McCalpin J. D., *Memory Bandwidth and Machine Balance in Current High Performance Computers*, in IEEE Technical Committee on Computer Architecture Newsletter, December, 1995

[McVo96] McVoy L., Staelin C., *lmbench: Portable Tools for Performance Analysis*, In Proceedings of the USENIX 1996 Technical Conference, pp. 279-294, San Diego CA, January 1996

[Mura89] Murata T., *Petri Nets: Properties, Analysis an Applications*, Invited Paper, Proceedings of the IEEE, Vol. 77, No. 4, April 1989

[Patt82] Patterson, D. A., Sequin, C. H., *A VLSI RISC*, Computer, **15**, No. 9, p. 8-21, 1982

[Pete81] Peterson J. L., *Petri Net Theory and the Modeling of Systems*, Prentice-Hall N.J, ISBN 0-13-661983-5, 1981

[SPAR90] SPARC International, *SPARC Architecture Manual, Version 9*, Mountain View CA., USA, 1990

[Ston87] Stone H. S., *High-performance Computer Architecture*, Addison-Wesley, 1987

Chapter 5

[CNCL] `ftp://ftp.comnets.rwth-aachen.de/pub/CNCL`

[Cede93] P. Cederqvist, *Version Management with CVS*, 1993

[Ferm97] Fermilab Physics Analysis Tools Group, *Histoscope Version 4.0 User's Guide*, FNAL SP0034, 1997

[RD3195] RD-31 Collaboration, *NEBULAS: High Performance Data-Driven Event Building Architectures Based on Asynchronous Self-Routing Packet Switching Networks*, CERN LHCC 95-47, Switzerland, 1995

[Steppl] Steppler M., Junius M., Görg C., *CNCL Manual*, RWTH, Aachen

[Teth95] Tether S., *A C++ Implementation of the RD-31 Generic Switching Fabric Simulation*, CMS TN/95 - 043, 1995

Chapter 6

[Alme95] Almesberger W., *High-Speed ATM Networking on Low-End Computer Systems*, Laboratoire de Réseaux de Communication, EPFL, Lausanne, August 1995

[Ande91] Anderson T. E., Levy H. M., Bershad B. N., Lazowska E. D., *The Interaction of Architectures and Operating System Design*, In Proceedings of 4th International Conference Architectural Support for Programming Languages and Operating Systems, ACM, Santa Clara CA, April 8-11, 1991

[Atki94] Atkinson, R., *Default IP MTU for use over ATM AAL5*, Internet Requests for Comments, RFC-1626, DDN Network Information Center, 1994

[Char97] Charlesworth A., et al., *Gigaplane-XB: Extending the Ultra Enterprise Family*, Symposium Record, Hot Interconnects V, Stanford University, Stanford CA, July 1997

[Chio96] Chiola G. Ciaccio G., *Operating System Support for Fast Communications in a Network of Workstations*, DISI-TR-96-12, DISI, Università di Genova, Italy, May, 1996

[Cull93] Culler D., et al., *LogP: Towards a Realistic Model of Parallel Computation*, In Proceedings of 4th ACM SIGPLAN Symposium on Principles and Practice of Prallel Programming, pp. 262-273, 1993

[Davie] Davie B. S., *Network-Friendly Workstations*, Bell Communications Research, New Jersey

[Davi91] Davie B. S., *A Host-Network Interface Architecture for ATM*, In Proceedings of SIGCOMM 1991, pp. 307-315, Zurich, Switzerland, September 4-6, 1991

[Davi93] Davie B. S., *The Architecture and Implementation of a High-Speed Host Interface*, IEEE Journal Selected Areas in Communications, Vol. 11, No. 2, pp. 173-180, February 1993

[Drus93] Druschel P., Abbott M. B., Pagels M. A., Peterson L. L., *Network Subsystem Design*, IEEE Network (Special Issue on End-System Support for High Speed Networks), 7(4), pp. 8-17, July 1993

[Drus94] Druschel P., Peterson L. L., Davie B. S., *Experiences with a High-Speed Network Adaptor: A Software Perspective*, In Proceedings of the ACM SIGCOMM'94 Symposium, pp. 2-13, London, September 1994

[Eick93] von Eicken T., *Active Messages: an Efficient Communication Architecture for Multiprocessors*, PhD. Thesis, University of California, Berkeley CA, November 1993

[Fouquet] Fouquet Y. A., et al., *ATM Performance Measurement: Throughput, Bottlenecks and Technology Barriers*, National Institute of Standards and Technology

[Hein93] Heinanen J., *Multiprotocol Encapsulation over ATM Adaptation Layer 5*, Internet Requests for Comments, RFC-1483, DDN Network Information Center, 1993

[Inter94] Interphase Corporation, *4615 SBus ATM Adapter Manual*, Dallas TX, USA, 1994

[Mand94] Mandjavidze I., *Software Protocols for Event Builder Switching Networks*, International Data Acquisition Conference, Fermilab, Batavia IL, October 26-28, 1994

[Mark97] Markatos E. P., Katevenis M. G. H., *User-Level DMA without Operating System Kernel Modification*, To appear in *Proceedings of the 3rd International Symposium on High Performance Computer Architecture* (HPCA-3), San Antonio, February 1-5, 1997

[McCa95] McCalpin J. D., *Memory Bandwidth and Machine Balance in Current High Performance Computers*, in IEEE Technical Committee on Computer Architecture Newsletter, December, 1995

[Oust90] Ousterhout J. K., *Why Aren't Operating Systems Getting Faster as Fast Hardware?*, In Proceedings of 1990 Summer USENIX Conference, Anaheim CA, June 11-15, 1990

[Ritc84] Ritchie, D. M., *A Stream Input-Output System*, AT&T Bell Laboratories Technical Journal, vol. 63, no. 8, pp. 1897-1910, 1984

[Stee94] Steenkiste P., *A systematic Approach to Host Interface Design for High-Speed Networking*, IEEE Computer, March 1994

[Sun95] Sun Microsystems Inc., *Thread Analyzer Answerbook*, SPARC Workshop 4, 1995

[Wulf95] Wulf W. A., McKee S. A., *Hitting the Memory Wall: Implications of the Obvious*, Computer Architecture News, 23, (1), pp. 20-24, March 1995

[Wels97] Welsh M., Basu A., von Eicken T., *ATM and Fast Ethernet Network Interfaces for User-Level Communication*, In *Proceedings of the 3rd International Symposium on High Performance Computer Architecture* (HPCA-3), San Antonio, February 1-5, 1997

Chapter 7

[Ande95] Anderson T. E., Culler D. E., Patterson D. A., et al., *A Case for NOW (Networks of Workstations)*, IEEE Micro, Feb, 1995

[Bask91] Baskett F., *Keynote Address*, International Symposium on Shared Memory Multiprocessing, Appril 1991

[Chio96] Chiola G. Ciaccio G., *Implementing a Low Cost, Low Latency Parallel Platform*, DISI, Università di Genova, Italy, November 1996

[Fing96] Fingerhut J. A., Jackson R., Suri S., Turner J. S., *Design of Nonblocking ATM Networks*, WUCS-96-03, Washington University, St. Louis, USA, 1996

[Gend97] Genda K., *TORUS: Terabit-per-Second ATM Switching System Architecture Based on Distributed Internal Speed-Up ATM Switch*, IEEE Journal on Selected Areas in Communications, Vol. 15, No. 5, June 1997

[Henn90] Hennesy J. L., Patterson D. A., *Computer Architecture a Quantitative Approach*, Morgan Kaufmann Publishers Inc., San Mateo CA., USA, 1990

[Mark96] Markatos E. P., Katevenis M. G. H., *Telegraphos: High-Performance Networking for Parallel Processing on Workstation Clusters*, In Proceedings of the High Performance Computer Architectures-2, San Diego, February 1996

[Myri95] Myricom Corporation, *MYRINET: A Gigabit per Second Local Area Network*, IEEE-Micro,Vol.15, No.1, February 1995, pp.29-36

[Nowa95] Nowatzyk A. G., Prucnal P. R., *Are Crossbars Really Dead? The Case for Optical Multiprocessor Interconnect Systems*, 22nd annual ACM International Symposium on Computer Architecture, June 1995

[Nowa95a] Nowatzyk A. G. Browne M. C., Kelly E. J., Parkin M., *S-Connect: from Network of Workstations to Supercomputer Performance*, 22nd annual ACM International Symposium on Computer Architecture, June 1995

[Part94] Partridge C., *Gigabit Networking*, Addison-Wesley, Professional Computing Series, ISBN-0-201-56333-9, 1994

[Part94a] Partridge C., et al., *Report of the ARPA/NSF Workshop on Research in Gigabit Networking*, Washington DC, July 20-21, 1994

[Witt92] Witte E. E., *A Quantitative Comparison of Architectures for ATM Switching Systems*, WUCS-91-47, Washington University, St. Louis, 1992

Curriculum Vitae

Objective

I am seeking career opportunities that provide me with creative challenges, while exposing me to a progressive development environment. I am also motivated to contribute to a team where my interest in new computer technologies and my skills in computer systems analysis, development and implementation will be effective assets.

Education

1993 - Present: Ph.D. in Computer Science, Technical University of Vienna, Austria

Subject: On a multiprocessor computer farm for on-line physics data processing

Emphasis on: data acquisition, computer architecture, system simulations, and computer systems performance evaluation.

1985 - 1991: M.Sc. in Nuclear and Particle Physics, Sofia State University, Bulgaria, *Date of Graduation:* November 1991.

Subject: On the experimental observation of Second Class Currents through the decay $\eta \rightarrow \pi e \nu$

Emphasis on: Standard Model, Cherenkov detectors, Monte-Carlo simulations, physics data analysis and statistics.

Skills

General Computer Knowledge:

Proficient with concurrent programming, high performance multiprocessor architectures, system simulations. Knowledgeable of operating systems internals, data communications and networking. Excellent understanding of object orientation concepts of modern high level programming languages.

Software and Hardware Knowledge:

Programming: C++, C, Java, Fortran, UNIX shell programming, TCP/IP, UDP/IP, RPC, LabView.

Operating Systems: In depth knowledge of Solaris (internals, device drivers, configuration, administration), and Linux. Working experience with IRIX, HPUX, AIX and Windows NT.

Platforms: SUN, Intel PC, Silicon Graphics, Hewlett-Packard, IBM RS6000, Apple Macintosh.

Experience

1997 - present: CERN Fellowship

CMS experiment, EP division, CERN - Switzerland

Responsibilities: Convene the CMS DAQ simulation group, high-level system administration, high-performance cluster architecture design and performance analysis. Provide support to a large user community.

1994 - 1997: CERN Doctoral Student

CMS experiment, ECP division, CERN - Switzerland

Responsibilities: Off-line simulation code maintenance and distribution, organized and administered the workstation cluster of CMS

1993 - 1994: Unix Consultant

Vienna Institute of High Energy Physics - HEPHY (took place at CERN)

Responsibilities: Code maintenance, software development

1991 - 1993: Physics tutor

Kalamata, Greece

Interests

Data acquisition, system simulations, computer architecture, operating systems, networks of computers, communications, parallel and distributed computing.

Personal Information

Date of Birth: 27 May 1965

Nationality: Hellenic

Marital Status: Single