FERMILAB-Conf-01/427 December 2002

Fault Tolerant Issues in the BTeV Trigger

presented by Joel N. Butler, for the BTeV Real Time Embedded Systems Group
whose initial members are
Jeffrey A. Appel, Tim Brennan, Joel N. Butler, Erik Gottschalk, Patricia McBride, Don Petravick, Ruth Pordes, Margaret Votava, *Fermi National Accelerator Laboratory*; Michael J. Haney, Ravishankar K. Iyer, Zbigniew T. Kalbarczyk, Mats Selen, *University of Illinois*; Daniel Kaplan, *Illinois Institute of Technology*; Daniel Mosse, *University of Pittsburgh*; Jae Oh, Sheldon Stone, *Syracuse University*; Ted Bapty, J. Fritz Barnes, Gabor Karsai, Sandeep Neema, Paul Sheldon, *Vanderbilt University*
(Dated: March 4, 2002)

The BTeV trigger performs sophisticated computations using large ensembles of FPGAs, DSPs, and conventional microprocessors. This system will have between 5,000 and 10,000 computing elements and many networks and data switches. While much attention has been devoted to developing efficient algorithms, the need for fault-tolerant, fault-adaptive, and flexible techniques and software to manage this huge computing platform has been identified as one of the most challenging aspects of this project. We describe the problem and offer an approach to solving it based on a distributed, hierarchical fault management system.

Introduction: The BTeV Trigger

BTeV is a dedicated B physics experiment designed to run at the Fermilab Tevatron collider [1]. The BTeV trigger will inspect all beam crossings, which occur at the rate of 7.6 MHz (every 132ns), for evidence of secondary decays of particles containing b-quarks. The trigger, which is described elsewhere in these proceedings [2], has three levels, all of which involve computation. The first level trigger performs track reconstruction, primary vertex finding, and impact parameter calculations, using data from BTeV's silicon pixel detector. There is also an alternative trigger path which uses data only from the BTeV muon system. A beam crossing contains on average two interactions. The required rejection at Level 1 is 99% and the efficiency is required to be greater than 50% for events that would survive all the final analysis cuts and show up as clean physics signals. We plan to use a system of 500 FPGA's and 2500 DSPs to do these calculations. While the trigger is making the Level 1 decision, all data for the crossing, which has been sparsified on the fly at an average rate of under 132 ns, is stored in a large, multi-terabyte buffer memory.

In order to maximize the utilization of all the processors, we have departed from many standard trigger practices that would cause inefficient synchronization points. For example, there is no fixed latency at any trigger level. Trigger decisions are communicated as soon as they are known and events, if accepted, are saved for subsequent processing or, if rejected, are erased and their memory is freed. Events are therefore not ordered after Level 1.

The second and third level triggers are performed on a 2500-5000 node LINUX farm. The algorithm uses all the detector elements to reconstruct the event with better resolution and additional information, including particle identification, to pick out events with potentially interesting final states or vertex topologies. The trigger does several different calculations so the final selection of events is not tuned only to specific states. This permits the trigger to select an eclectic mix of analyzable decays involving the b-quark. It is highly efficient for states we believe to be interesting today but also has reasonable efficiency for nearly any kind of B decay, including those that may become interesting in the future. Level 2 and 3 together produce an additional rejection of 95% of the triggers surviving Level 1 and have a combined efficiency of more than 85% for most B decays of potential interest. Approximately 4000 events per second will pass the trigger (this number includes signal, trigger failures, calibration events, special triggers for charm studies, a prescaled sample of unbiased events, and a contingency factor).

I. THE PROBLEM

This system must operate with excellent reliability. If the system is down, data will be lost and the science will suffer. However, with so many components, something is always likely to be broken and it is necessary for the system to run as efficiently as possible with faults present. Since failures will occur in operation, it will be necessary to fix problems on the fly or notify human operators.

Eventually, all the data will be analyzed and physics will be extracted. In doing the analysis, it is necessary to understand what problems existed during data taking, so faults and any actions taken in connection with them will need to be reported and logged. Also, the trigger is connected to a real experiment and a real accelerator

and there will be changes in the machine operating conditions, the detector itself will experience problems, and the environment (temperature, humidity) will change. Since all data passes through the trigger, it makes sense for the trigger algorithms to monitor the health of the experiment and to "reuse" many aspects of the fault management system, for example the fault logging and notification processes, to track and even to fix detector problems or to make needed adjustments over the course of a data run.

We also recognize that the "trigger processor" complex will need to be used very flexibly for debugging and commissioning the detector itself. We will need to use it to carry out tests of new trigger hardware and to validate new trigger algorithms before going into full operation with them. We would like to be able to use idle cycles, either when the accelerator is not operating or when there is spare capacity during operation, to carry out other compute-intensive tasks, such as detector simulation, to get efficiencies, or detailed analysis of calibration runs.

A review committee [3] set up by Fermilab Management made the following assessment:

"...Given the very complex nature of this system where thousands of events are simultaneously and asynchronously cooking, issues of data integrity, robustness, and monitoring are critically important and have the capacity to cripple a design if not dealt with at the outset. It is simply a fact of life that processors and processes die and get corrupted, sometimes in subtle ways. BTeV has allocated some resources for control and monitoring, but our assessment is that the current allocation of resources will be insufficient to supply the necessary level of 'self-awareness' in the trigger system. Without an increased pool of design skills and experience to draw from and thermalize with, the project will remain at risk."

We have translated this into the following requirements.

- The system must be highly available, since the detector produces data continuously over a long period of time.
- To achieve high availability, the system must be fault tolerant, self-aware, and fault adaptive.
- Faults must be corrected in the shortest possible time, and corrected semi-autonomously (i.e. with as little human intervention as possible). Hence distributed and hierarchical monitoring and control are vital.
- The trigger system must be dynamically reconfigurable, to allow a maximum amount of performance to be delivered from the available, and potentially changing resources.
- The trigger system must be capable of expansion to at least a factor of three more nodes to permit more sophsiticated computations, operation at higher luminosities, or addition of algorithms to pursue other physics topics.
- The system must have excellent life-cycle maintainability and evolvability to deal with new trigger algorithms, new hardware and new versions of the operating system.

There are also some special requirements related to the "flexibility" of the system. We want to be able to dynamically allocate portions of the system to testing new algorithms or hardware. We want to be able to dynamically allocate portions of the L2/L3 farm to reconstruction and analysis (there will be a huge amount of disk on the system to retain data for months) or to simulations. We want the system to be easily able to change modes during a "store" from alignment to normal operation and also to running special diagnostics for the detector. We may want to dynamically adjust the "mix" of accepted triggers.

II. THE PROPOSED SOLUTION

The proposed solution is a hierarchical, distributed fault management system which has a two-tier "System Design and Run Time Framework," shown in Fig. 1. The two main tiers are discussed below. The Run Time Framework has its own sub-hierarchy of distributed fault managers.

A. The Design and Analysis Environment

This tier of the system, which is above the dotted line in Fig. 1, carries out the following functions:

- Modeling
  - Information/Algorithm Data Flow Modeling
  - Target Hardware Resource Modeling

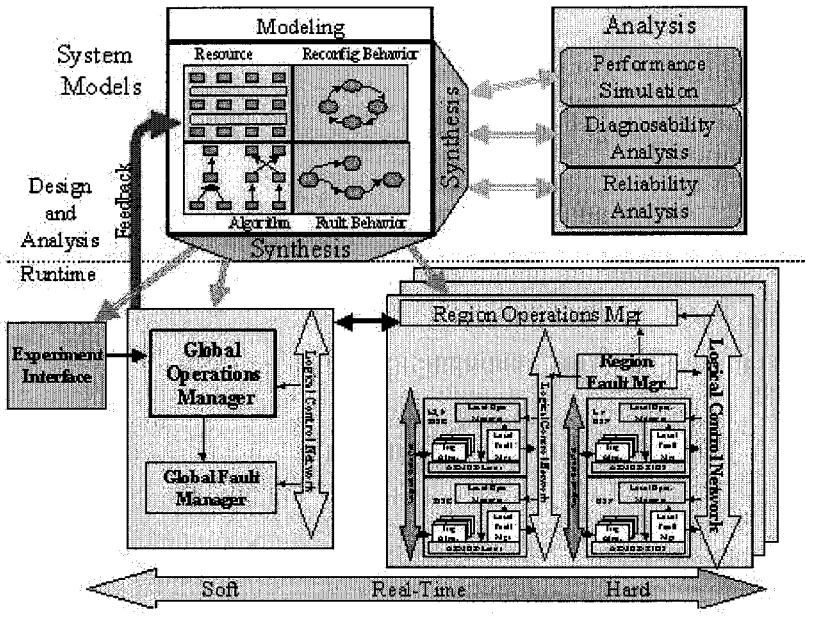

FIG. 1: Bi-Level System Design and Runtime Framework- System Models use domain-specific, multi-view representation formalisms to define system behavior, function, performance, fault interactions, and target hardware. Analysis tools evaluate predicted performance to guide designers prior to system implementation. Synthesis tools generate system configurations directly from models. A fault-detecting, failure-mitigating runtime environment executes these configurations in a real-time, high performance, distributed, heterogeneous target platform, with built-in model-configured fault mitigation. Local, regional, and global perspectives are indicated. On-line cooperation between runtime and modeling/synthesis environment permits global system reconfiguration in extreme-failure conditions.

  - System Detection and Fault Mitigation Modeling
  - System Constraint Modeling
- Analysis
- Synthesis
  - Design Alternative Resolution, Partitioning and Processor Allocation
  - System Configuration Generation
  - Operation and Fault Manager creation and configuration

The approach adopted for this part of the system is called Model Integrated Computing [4]. It will be used both as a design and simulation tool and for analysis and fault modeling during running. In the design phase, it will permit modeling of the system performance especially as relates to fault tolerance. It will be used to study key issues such as how the run time fault management hierarchy should be configured, what the impact of various fault conditions will have on system performance, etc. It will also have the ability to generate and download the various systems that it models onto the actual hardware.

During operation, it can continue to model the system and, if a problem arises, it can attempt to simulate the problem and provide operators with a modeling and analysis tool for diagnosing and remediating it. It can also generate a new system configuration and download it onto the hardware.

B. Run Time Environment

The Run Time Environment is responsible for providing the fault tolerant, fault adaptive software for each resource in the system, including computers and their programs and network segments and switches. It is a distributed, hierarchical system which provides for local, regional (probably several levels), and global fault managers. When it detects a problem, a local fault manager will try to respond. If it cannot, it passes the problem on to a higher level in the hierarchy. Information on all fault conditions and actions taken eventually percolates to the global level where they are logged for use in fault trend analysis and reliability analysis and in interpreting the event data offline.

The components of the Run Time system:

- Operating system for DSPs and INTEL UNIX systems
- Run Time Hierarchy
  - Very Lightweight Agents (VLAs), which are simple software entities that expose errors in DSP kernel behavior
  - Adaptive, Reconfigurable and Mobile Objects for Reliability (ARMORs), which are responsible for fault handling at the process level
  - Hierarchical Detection and Recovery
    - Node level
    - Regional level(s)
    - Global level
    - Feedback to/from the modeling environment
- System Validation through software based fault injection
- Collection of data for creating and validating new fault models

The lowest level of the Run Time Hierarchy is shown in Fig. 2. An example of actions that might be taken in the case a hung DSP are to reset the DSP and restart the program. If this happens too frequently, the fault could get passed up to a higher level, which would check whether there were any correlations with other system behavior. If not it might conclude that this was a hardware problem and could take the DSP out of operation. Or it could conclude that a pattern was developing that was indicative of a more global problem.

C. Other Aspects of the System: Interfaces

The system must provide several other capabilities and interfaces, which include: 1) Run Management/Control; 2) Persistent Storage (resource management, run history). Although faults may be handled at any level of the hierarchy, fault and mitigation data are always passed to the highest level so that the experiment can track all conditions affecting the data; 3) User interface/diagnostics/automatic problem notification; and 4) Application code. The application code and physics algorithms will use the same underlying infrastructure of the fault tolerance system. ARMOR has an API for this purpose.

The API can be used by "physics programs" in the trigger to report problems detected in the data stream in the same manner that trigger hardware problems are reported. In the simplest case, this would mean using the same reporting and logging mechanism. However, it may be possible to supply fault mitigation as well by manipulating trigger parameters or even the slow control system and these would use strategies similar to the Run Time fault management system. If this can be implemented, the trigger system would become effectively the "watchdog" for the entire trigger, detector, and accelerator operation.

III. STATUS AND CONCLUSION

Providing fault tolerance may be the most difficult issue in successful implementation of the trigger. BTeV has an architecture, a plan, and a project to produce a fault tolerant, fault adaptive system. The work done on this project should have wide applicability to large parallel systems with very high availability requirements.

This project has received a grant from the National Science Foundation's Information Technology Research (ITR) program [5].

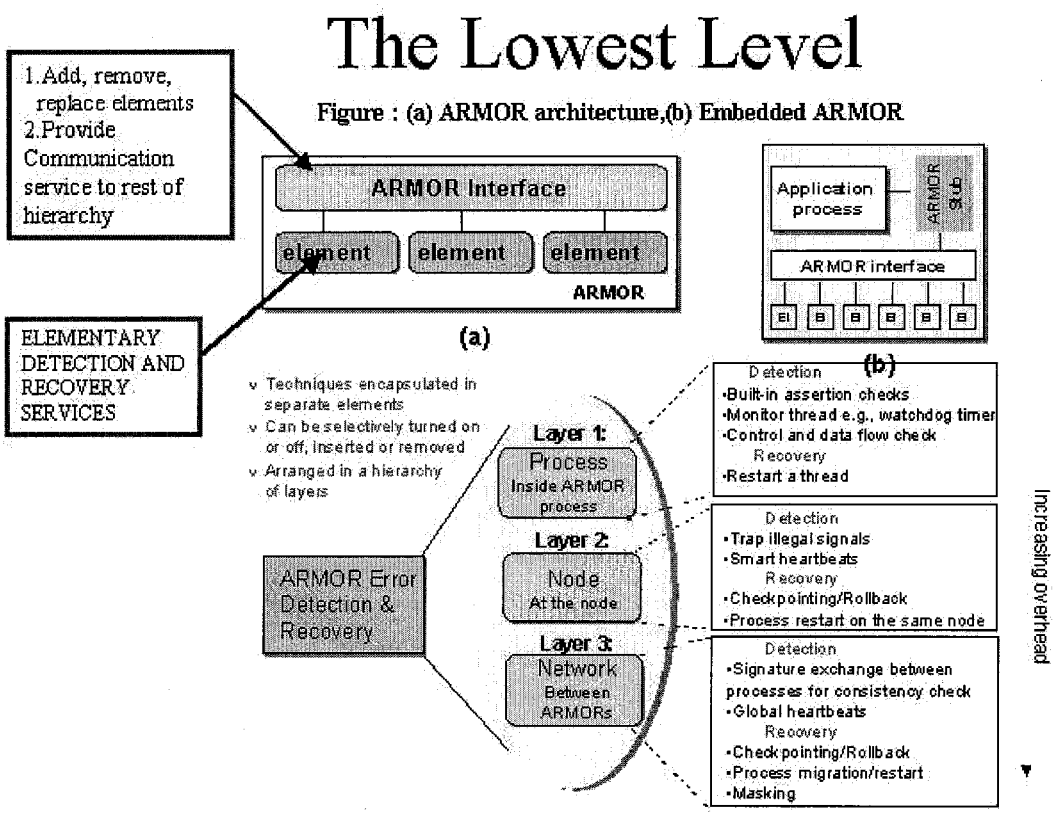

FIG. 2: (a) ARMOR architecture; (b) Embedded Armor; (c) and the ARMOR Error Detection and Recovery Hierarchy.

Acknowledgments

We wish to acknowledge the help of the BTeV trigger group and the BTeV Data Acquistion Group in developing this approach.

---

[1] The BTeV Proposal (May 2000) resides at:
http://www-btev.fnal.gov/public_documents/btev_proposal/index.html
The BTeV trigger is covered in chapters 9, 10, and 14.

[2] Dan Kaplan, A BTeV Vertex Trigger System, these proceedings.

[3] Robert Tschirhart and Peter Wilson, Private communication.

[4] Frank, H., Sztipanovits, J., and Karsai, G., "Model-Integrated Computing," *Proceedings of the 1997 Hawaii Systems Sciences Conference* (CD ROM publication), 1997 and
Sztipanovits, J., et al.,"MULTIGRAPH: An Architecture for Model-Integrated Computing, *Proceedings of the IEEE ICECCS '95*, pp.361-368, November 1995.

[5] NSF Award #ACI-0121658