Scientific Computing Autonomic Reliability Framework

Abhishek Dubey, Sandeep Neema
Institute for Software Integrated Systems
Vanderbilt University, Nashville, TN

Jim Kowalkowski, Amitoj Singh
Fermi National Accelerator Laboratory
Batavia, IL

***Abstract*— Large scientific computing clusters require a distributed dependability subsystem that can provide fault isolation and recovery and is capable of learning and predicting failures, to improve the reliability of scientific workflows. In this paper, we outline the key ideas in the design of a Scientific Computing Autonomic Reliability Framework (SCARF) for large computing clusters used in the Lattice Quantum Chromo Dynamics project at Fermi Lab.**

I. Problem Motivation

Data processing within the Large Quantum Chromo Dynamics (LQCD) project at Fermi Lab is carried as analysis campaigns (scientific workflows), which consist of an input dataset and a set of interdependent processing steps (called jobs) executed over large commodity computer clusters. These large clusters built with commodity computers yield the highest performance per dollar but exhibit intermittent faults, which can result in systemic failures when operated over a long continuous period for executing analysis campaigns.

Typical running time of a campaign may span several months and execute several hundreds of data and computation intensive, parallel, MPI Jobs, requiring several processing nodes over its lifetime. Typically, several users analyze different workflows on the clusters concurrently. Diagnosing job problems and failures in this complex environment is an arduous task, specifically when the success of whole campaign might be affected by even one job failure. Table I summarizes some common faults leading to failure of the workflow as experienced in the LQCD clusters.

Manual administration though essential is slow to respond to the intermittent faults. Therefore we need to supplement it by an autonomic management subsystem that can provide: (a) **Fault Prediction**: Identify the trends that might lead towards faults, (b) **Fault Isolation**: Identify the source of fault, (c) **Fault Mitigation**: Ensure that the resources of the cluster are used to the best possible extent and achieve the best possible start to completion ratio of jobs, even in the presence of hardware/software failures.

Generalized frameworks have been used in the past to monitor the health and status of cluster resources. Early tools such as ClusterProbe concentrated on per node monitoring and visualization without considering the health of the cluster as a whole. Other tools such as NetLogger, Monitoring and Managing Multiple Clusters (M3C), and Java Agents for Monitoring and Management (JAMM) were tailored for centralized, human-in-the-loop management of clusters but little investigation was done to provide autonomic monitoring and control of cluster computing environments. Other recent frameworks such as Ganglia and Nagios have been developed in last five years. These frameworks are extensible and even provide for simple centralized control. However, all of these frameworks lack a formal design methodology behind it.

TABLE I
COMMON FAULTS → RESULTING FAILURES

Fault	Failure
Communication Link Errors	Job Termination
Storage Full	Job Termination, Job Hang
CPU Temperature High	Node Shutdown, Job Termination
Out of Memory	Job Termination
Unexplained Node Restart	Job Termination

Model-based design, is one such formal system design methodology that has gained momentum in recent years as a sound methodology of applying computer-based modeling and synthesis methods to a variety of problem domains, including distributed systems. A benefit of using formal models is that they can be queried or transformed to produce a variety of domain specific artifacts, which are critical to deployment and execution of the system, but are tedious and error-prone to produce manually.

II. Overview of SCARF

Fig. 1 describes the architecture of *Scientific Computing Autonomic Reliability Framework (SCARF)*, being developed by our research group for the LQCD computing clusters. The basic components of this framework are distributed monitoring units, fault-mitigation units and a system wide planner for dealing with resource reallocation in case of severe failures. It is a Reflex and Healing (RH) architecture [1] (and references therein), which consist of hierarchical network of decentralized fault management entities, called reflex engines that are instantiated as state machines or timed automatons that change state and initiate fast reflexive mitigation actions upon occurrence of certain faults.

Centralized Control: The centralized controller (Healing) consists of a design and modeling environment, an inference engine to infer the current system health and a planner portion that can be used to alter the allocation of system resources to jobs in order to mitigate failures.

The design and modeling environment provides for specification of different resources in the system (computation nodes). These resources are annotated with the parameters such as CPU utilization that uniquely identify the state of that

978-0-7695-3535-7/08 $25.00 © 2008 IEEE
DOI 10.1109/eScience.2008.113

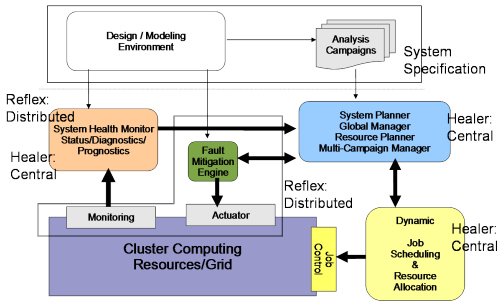

Fig. 1. Architecture: Annotations identify the centralized and de-centralized components

machine. These state variables are dynamically updated to reflect the current state of the cluster. A central visual interface can be constructed to provide a compact representation of system state to an administrator. SCARF also provides a modeling language for specification of workflows as directed acyclic graphs by using "jobs" that are specified in a library. A job can be reused between workflows and also the historic performance of a job can be monitored centrally. Lastly, the modeling environment consists of various constraints that can be used to allocate jobs in workflows to computation nodes, reallocate resources in case of failure and retire workflows that cannot be completed in a timely manner.

Statistical Analysis: We are working towards developing a light-weight, first-order diagnoser that can predict some of the more frequent faults. The diagnoser would be flexibly composed of a set of analyzers that reason with system health parameters. For example, the following analyzers can predict imminent job failures:

- We are developing a correlation model between CPU utilization and temperature for a class of machines. CPU temperature that is 2 standard deviations away from the predicted mean for the current CPU utilization is a reliable predictor of imminent node and job failure.
- We are developing Disk utilization profiles for a class of jobs. This profile can predict how much space is going to be needed as the job progresses, and in case of insufficient space, it can be used to proactively alert the administrator.
- We are developing correlation model between important predictor variables such as CPU fan speed and CPU temperature. Out of range observations for these variables over a significant period of time, are also reliable predictor of imminent node and job failure, and can be used to proactively alert the administrator.

Distributed Runtime: The run-time portion of the framework is distributed across the system. Each cluster computing node runs an agent called "manager" (fig. 2). These managers are an instance of a reflex engine [1] and oversee the sensors and actuators scripts needed to monitor and actuate that particular machine. These scripts are downloaded on the particular node based on the specification provided in the modeling environment. The actuator scripts present in a manager are executed based on various fault mitigation strategies downloaded on that node. The execution sequence of sensor and actuator scripts is governed by the manager's scheduler, which enables synchronized execution and reduces jitter between several periodic executions of sensors on a given node [2].

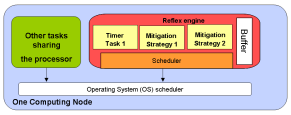

Fig. 2. A Reflex engine

To reduce complexity, the managers are divided into regions based on their racks (fig 3). Each region is managed by a head node that is identified as the regional manager. This manager relays the sensor information for the computing nodes under its supervision to the database. It is also responsible for the general health of nodes running under its supervision. For that purpose, it keeps a running moving average of critical parameters such as CPU utilization, and CPU temperature, for the nodes under its supervision.

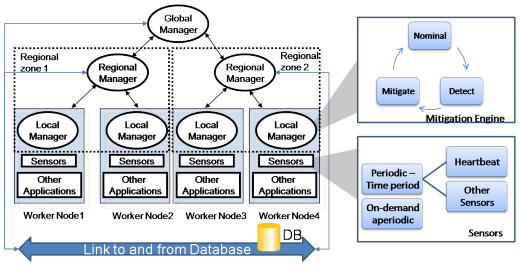

Fig. 3. Hierarchical reflex engines

III. Conclusion

We have described a model-based reliability framework for computing clusters in this poster. This framework enables the specification of workflows, mitigation strategies and the state variables to monitor in the system. Currently, we have deployed the runtime framework on three LQCD clusters. The sensor data that we have collected so far provides us the ability to perform analysis in response to a failure, as well as develop the correlation models and nominal behavior models that can be used in the analyzer. We are currently working towards design and deployment of mitigation strategies that can actuate changes in a computing node from the respective local manager. We have developed and presented the workflow modeling language and a fault recovery algorithm earlier in [3].

References

[1] Dubey, A. et al. Towards a verifiable real-time, autonomic, fault mitigation framework for large scale real-time systems. *Innovations in Systems and Software Engineering 3* (March 2007), 33–52.

[2] Dubey, A. et al. Towards a model-based autonomic reliability framework for computing clusters. In *EASE '08* (2008), pp. 75–85.

[3] Nordstrom, S. et al. Model predictive analysis for autonomic workflow management in large-scale scientific computing environments. In *EASE '07* (2007), pp. 37–42.