Aids to Pro-active Management of Distributed Resources through Dynamic Fault-Localization and Availability Prognosis

***Abstract*—One aspect of running applications over distributed resources is susceptibility to failures like system crashes, memory leaks, network congestions, overloaded resources, etc. Managing applications in face of such failures is a complex task. This is prominent in Grid like environments, wherein geographical diversity with resource heterogeneity is an inherent characteristic. An approach to masking of resource failures is to build applications that can recognize the failures and recover from them. This burdens the developer with the complex task of understanding and recognizing resource failures. This could be a daunting task, particularly with heterogeneous resources. The other approach is that the system recognizes the effect of resource failures over corresponding applications and takes appropriate actions to ensure transparency to these failures. Problem with this approach is characterizing the resource failure effect on applications.**

In this paper we propose a middle-path approach that uses both sides information to effectively mask resource failures in distributed environments. We describe a framework that illustrates the context information needed within a distributed setup, to enable pro-active management. We argue that if the system publishes its resources' availability information, the application can use the information to recognize resource failures and adapt to them. For this, the system should be capable of detecting its internal failures and do fault-localization at runtime, and publish this fault effect as a loss in availability of the resources. As a first step towards this, we detail a dynamic fault localization method to obtain the availability information of the system hardware layer at runtime. Using the current availability information we make availability prognosis. The initial studies made by us indicate that the predictions are 90% of the time correct and show deviations in about 10% of the cases. This indicates the usability of these predictions at the time of scheduling the application.

***Index Terms*— availability prognosis, distributed systems, fault-localization, resource failures.**

I. Introduction

DISTRIBUTED systems are a collection of loosely coupled resources having independent control. Each system has its own operating system that manages its local resources. An application intending to use distributed resources has to address the lack of central control over resources particularly with respect to resource failures. Detecting and managing resource failures in distributed environments is a complex task in itself. And this complexity increases many folds in Grid [1] like environments, wherein geographical diversity with resource heterogeneity and independent administrative domains, is an inherent characteristic. In such setups, an application intending to use multiple systems does not have uniform access to information about all resource failures that could potentially affect it. In the present scenario of distributed environments, this is addressed as a best effort case. In existing schemes, on one hand, an application is written to trap elaborate exception conditions and make a best guess of what could have gone wrong. On the other hand, the management services monitor applications periodically, to verify its execution status using the local system's process status. There are examples of end-to-end QoS provisioning, wherein the application requirements are guaranteed within the perspective of performance factors, but not in face of resource failures.

Many times, in large distributed setups, there is enough resource redundancy that could have easily fulfilled the applications requirements. The lacuna lies in the fact, that at runtime, for an application, the resource failure or an impending resource failure is not visible, even though the system is aware of it. And, it is not possible for a system to predict when an application would use the failing resource. The application notices the resource failure when it's usage request fails. At this point the application may wait forever or fail totally. In either case, the application needs user intervention. This could mean loss of time and productivity for the user.

With the advent of Service Oriented Architecture (SOA), [2]-[4] applications and resources in the distributed environment are viewed as extensible set of services. SOA provides an organized mechanism for representing a service with its properties, state and functionality. Within SOA, user requests are fulfilled by coordinated interactions among various services (e.g. Open Grid Services Architecture - OGSA). In such architectures, each service maintains its state or context-information as a property, which can be accessed by an application intending to use the service. In this paper, we use Grid as a representative example of large-scale distributed system and explore mechanisms to provide proactive fault management of applications, with specific focus on highly available applications. To render high availability within such architectures, we propose that if each interacting service, publishes it's availability information, the management module monitoring the application using that service can monitor the service's availability information for adapting to failures by noticing changes in this information. Proactive management services also benefit if this availability information is predicted for a future time. Proactive management can be effective if one can identify or predict failures ahead in time when mechanisms to mask failures can provide adaptivity. Of course these mechanisms are rendered useless in case of sudden failures where only reactive approaches offer solution.

In order to enable proactive fault adaptivity to failures of highly available applications we need to have the following:

1. Runtime Physical Resource Availability Information.
2. Impact of physical resource and system software failures on the high available process in consideration.
3. Failure Monitoring and adaptivity services.

Resource Availability Information: Every service is bound to a host in the distributed environment. To arrive at a meaningful availability number, for the service, apart from considering the service as a software (software availability) we also need to consider the current availability of the physical resource the service is running on. For this, the physical resource has to recognize it's internal component failures at runtime and expose the failures in a standardized, resource-independent manner. Such a standardized, resource-independent representation of the availability of various hardware components, comprising a system, is included in the system's hardware health vector. In this paper we use hardware health vector and resource availability information interchangeably.

Impact of physical resource and system software failures: System software (operating system and distributed middleware) need to map hardware resource failures to the corresponding high level resources like virtual memory pages, files, processes, threads, sockets, etc. being used by an application process, of interest, and convert the hardware health vector to the process health vector. The process health vector contains not only the software availability of the process but also the availability of the virtual resources (operating system's abstractions of the hardware devices) the process is using or intending to use.

Failure Monitoring and adaptivity services: Management services enable monitoring and autonomic application recovery in face of failures. Examples of these are adaptive schedulers that take scheduling as well as rescheduling decisions based on predicted and current availability information.

In this paper we describe a framework that enables a distributed application to autonomously adapt to resource failures. At present, for the context of this paper, we consider only hardware component failures impacting the application under consideration. We expect to extend this to include the impact of system software failures also. At each level of the framework we detail the function of the components and the necessary availability information required for that functionality. Following this, we state how a distributed scheduler could use the resource availability information to enable application adaptivity to hardware failures. As a first step to providing resource availability information, we describe a method to dynamically localize hardware faults. The identified failure is exposed as health vector of hardware resources representing the current working probability of each resource. We also state a method to compute the expected availability, in future time, of a hardware resource based on its current availability.

II. Review of Related Work

Based on the contributions of the paper, the review of related work is divided into three sub-sections, namely:

A. Review of pro-active fault management frameworks for distributed systems.
B. Review of resource availability information representation.
C. Review of fault localization methods.

A. Framework for pro-active fault management in distributed environments.

Most techniques on QoS management and fault tolerance, reported so far, either satisfy QoS guarantees or handle failures; each is addressed in isolation to the other. In large scale distributed environments like the Grid, where applications could run for many days or use services that are not locally accessible, it is necessary to address these two issues in conjunction. There are very few references on satisfying QoS in face of hardware failures and hardly any for proactive fault management for distributed systems. Most are reactive fault tolerant methods that use replication to overcome effects of faults. In this context there are various methods reported to correctly and efficiently communicate (group communication protocols) failed resource information for taking a decision on consistency of results. One reported work that brings out the issue of addressing real time guarantees in face of failures is "MEAD: Support for Real-Time Fault-Tolerant CORBA"[5]. The MEAD system captures the need for considering the effect of faults/restarts on real-time behavior, uses replication of scheduling and resource management components and schedules recovery to avoid missing deadlines. The MEAD system addresses meeting timing guarantees in real-time systems in face of failures. In real-time systems the deadlines are stringent, which may not be directly applicable to Grid environments. In Grid environments the QoS guarantees need to be treated as soft real time deadlines. Our framework considers combining the issues of guaranteeing QoS requirements with fault tolerance by taking advantage of potentially abundant resources for recovery of applications in face of failures. We apply the layered recovery strategies taking advantage of multi-tier architecture of Grid.

One recent paper describes a pro-active fault-tolerant framework for Grid [6]. This paper presents an agent oriented, fault tolerant grid framework where agents deal with individual faults proactively. An agent is a software service that can take decision on behalf of a user or an application and initiate an action. In this framework the agents maintain information about hardware conditions, executing process' memory consumption, available resources, network conditions and component mean time to failure. Based on this information and critical states, agents strive to improve the reliability and efficiency of grid services. The information of the system that the agent collects is the Mean Time To Fail (MTTF) and Mean Time Between Failures (MTBF) values along-with the system age. MTTF and MTBF are average values and hence are coarse-grained for predicting real time behavior. Also, deriving these values for a system typically needs a statistical study with large data collection. Age of a system typically depends on the usage of its components and not just the time since its purchase. Any method using these numbers may lead to under utilization of the system even when it has capability to deliver correct functionality.

In this paper, we propose a framework that uses the availability information of service for proactive fault management in a layered approach. This approach is different from others cited above as it uses the current availability information that is based on observing the current state of the resources used. The availability information used is fine-grained and gives the true picture of resource failures in the system.

B. Resource availability information

Availability is a property attributed to a system and understood as its readiness to perform a designated service. This readiness is expressed by the MTBF value. MTBF represents a statistical average and requires collection of large data over long periods of time to arrive at a meaningful value. In this paper we propose a novel method to represent resource availability. Any computing system consists of hardware components that are composed according to the system's architecture for a desired functional specification, and is wrapped around in system software to provide the necessary user interface. In order to draw conclusions on a system's availability we need to consider the current state of the system hardware as well as its impact on software with respect to the failures that are occurring. We need a method to identify this current state and represent the system availability as a composite of its components' and software availability. In this paper we propose a method to identify a system's current hardware failures and analyze the effect of these failures, at runtime, from the perspective of its functionality and publish them as a resource availability vector. We call this method dynamic hardware fault localization. The method gives a simple mechanism to quantify and represent hardware resource availability information. As of this study we believe that there is no published reference on such computation or representation of resource availability information.

C. Dynamic Fault Localization

Many research efforts have been reported on analyzing event logs [7]-[13]. Significant techniques that standout in these analyses are statistical fits to log data [7], factor and cluster analysis [8]and tupling schemes [9]-[10]. In [7], the authors discuss methods for statistical analysis of error logs for modeling intermittent and transient errors individually and together. They report that since error logs contain an intermix of error events from a number of error sources, it is essential to separate related errors based on the error sources, to reach a statistically meaningful conclusion. The authors used Dispersion Frame Technique (DFT), based on heuristics, to separate error logs into their constituent error sources and provide a set of rules for fault prediction. DFT is reported to have predicted 93.7% of the hard failures. In [8] raw event log data is reduced to manageable clusters using event filtering and time domain clustering methods. Probability distributions characterizing the errors were obtained based on 7-month logged data. Multivariate statistical techniques are used to correlate error and failure dependency among different system components. The tupling methods reported in [9]-[10] focus on generating appropriate rules for data reduction to segregate repeated or related events. This work examined large sets of data collected over extended period of time to establish these rules. Recent publications [11]-[12] report the use of time-series algorithms for predicting errors. The authors of [11] report the evaluation of three different algorithms, namely, time-series, rule-based classification and Bayesian network algorithms, for event analysis. For each of these algorithms, the parameters necessary for modeling are generated out of data collected over a period of one year. The authors combine time-series analysis and rule-based classification algorithms with the Bayesian network analysis for predicting future events based on the observed ones.

Yet another recently published work on predicting failures using Hidden Markov Models [13], also extracts failure patterns from log collections. It is noteworthy to observe here that in all the approaches reported above, error log data collected over long period of time was statistically studied and methods are evolved based on these studies. The approach proposed in this paper is non-statistical method and hence does not depend on such a study. Instead, the method uses the system architecture for deriving possible failure patterns and encodes this into a Bayesian Belief Network in terms of independent and dependent failure probabilities of components in the system. The errors being logged are dynamically segregated into related groups based on spatial and temporal classification rules and marked as evidences on the Belief network. The inference derived out of the Belief network is used to localize the fault to the respective hardware component. So far reported literature indicates studies done on massive collections of error event logs. We, in this paper, use a method that works on runtime error events being generated in the system and derive all the parameters for analyzing these events from the system architecture. To the best of our knowledge there exists no earlier work based on this approach.

III. Frame work for Pro-active Fault Management

In this section we detail the framework for pro-active fault management for an application in distributed environments like the Grid. The framework is however not restricted to Grids. Figure 1 shows a schematic diagram of the framework. The diagram depicts a typical multi-tier architecture of distributed environments. In this figure there are four-layers, namely, Grid user integrated development environment (Grid IDE), Adaptive Grid Meta-Scheduler (AGMETS), Grid Middleware and the Grid Fabric, in order of flow of execution control. A Grid user composes his application and states it as a workflow composed of job steps using the Grid IDE. Each job step could be an independent job composed of collaborating services. A job step is specified with its resource and QoS requirements. Resource availability is specified by the job step as a QoS requirement. A highly available application would specify its availability requirement as 100% resource availability. For example, the job step specifies its resources requirements as:

```
<ncpus=1,memory=1Gb,input_file="/home/myinput",outpu
t_file="/homr/myouput",scratch_dir="/home/scratch">
```

and required resource availability as:

```
  <processor=1.0,    memory=1.0,    input_file=1.0,
output_file=1.0, scratch_dir=1.0>
```

The value of 1.0 as availability number against a resource means that the application expects all its required resources to be 100% available, at all times, during its execution run. An application that wants a best of effort run need not specify availability as a QoS requirement. AGMETS, Grid Middleware and the Grid Fabric use the availability information specified by the job step as an input for making scheduling, monitoring and recovery decisions. After the user completes the composition of his application, the Workflow manager calculates each job step's dependencies, composes an execution plan and submits each job step according to this plan to AGMETS [14]-[15]. AGMETS parses the resource and availability requirements of the application and selects a suitable compute node for dispatching the job. AGMETS uses the job submission service of the Grid Middleware for this. Solid arrows depict this flow of execution control in Figure 1. Once the job step is in execution, AGMETS starts monitoring the host's resource availability against the QoS requirements specified for the job step. AGMETS gets this information from the Grid Middleware's resource service provider. Dashed arrows in Figure 1 depict this flow of information.

The job submission service of the Grid middleware sets up the Grid Job Service Monitor and Manager services after the job has been successfully submitted to the compute node. If the current host's resource availability drops below acceptable limit, AGMETS triggers job-step recovery action like issuing an application checkpoint and job step migration to the next suitable node. The Grid Middleware publishes the host's resource availability information specific to the job step, the process health vector. The Job service monitor uses this information to watch current availability of resources for the job step. In case a specific resource's availability drops below acceptable limits, the monitor triggers resource specific recovery action. For example, the compute node detects file read error on the job step's input file, after the time it has started reading, due to media errors on the file staging disk. At this point in time the job-step would fail with an I/O error on input_file. Since now the compute node is publishing loss of availability on the input_file, the Grid Job Service Manager can take an autonomous decision on relocating the file by issuing GridFTP afresh. The process resource manager running on compute node would request the operating system of the compute node to relocate the file from the recent transfer to an area on disk that has so far not experienced any media errors or locate it to an alternate disk.

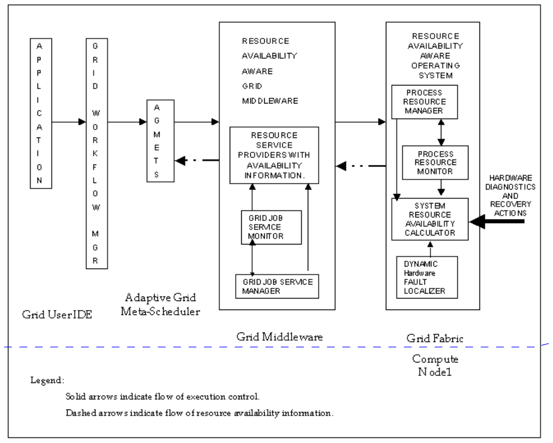
APPLICATION
GRID WORKFLOW MGR
AGMETS
RESOURCE AVAILABILITY AWARE GRID MIDDLEWARE
RESOURCE SERVICE PROVIDERS WITH AVAILABILITY INFORMATION.
GRID JOB SERVICE MONITOR
GRID JOB SERVICE MANAGER
RESOURCE AVAILABILITY AWARE OPERATING SYSTEM
PROCESS RESOURCE MANAGER
PROCESS RESOURCE MONITOR
SYSTEM RESOURCE AVAILABILITY CALCULATOR
DYNAMIC Hardware FAULT LOCALIZER
HARDWARE DIAGNOSTICS AND RECOVERY ACTIONS
Grid UserIDE
Adaptive Grid Meta-Scheduler
Grid Middleware
Grid Fabric
Compute Node1
Legend:
Solid arrows indicate flow of execution control.
Dashed arrows indicate flow of resource availability information.

Figure 1: Pro-Active Fault Management Framework for Distributed Environments.

Comment [k1]: Layers would require that the boxes are stacked up one above the other. But here they are vertical boxes. So the notion of layers is not obvious.

In this framework, we use the concept of recovery oriented computing [16] to incorporate autonomous behavior, in face of failures, to the application. The necessary information to take decisions on when to start recovery is provided by the health vectors. At each layer of the above framework, we define a layer-specific health vector associated with the job-step under observation. For example, AGMETS looks at the compute node's health viz. Node1's current capability to accept or complete the job-step; the Grid middleware looks at health of a specific resource used by the job, for example, health of input file for the job-step in the process health vector; and the compute node looks at process specific resources viz. health of the disk blocks and the file-system cache buffer pages associated with the input file. We envisage these health vectors to be interdependent and composed at runtime. The interdependence is bottom-up, i.e., once we know the compute node specific hardware resources (like disk, processor, etc.) availability, we can map the process' resources (like thread, file, stack, etc.) health to that of the hosting hardware resources and from these compute the job-step's host's health.

As of this study, we have computed the health vector for the hardware resources of a node like health of the disk, processor, memory banks, etc. The definition and computation for the process and job related health, are underway. In the framework described here we use a dynamic method for hardware fault localization to compute the hardware health vector, which is described in section IV. The method analyses runtime error events generated on the system to compute the current resource availability information. The method also makes an availability prediction based on the observed error events. The resource managers and the schedulers can use this predicted resource availability while making resource reservations or allocations. In the following section we describe the dynamic fault localization method with availability prognosis, and show initial results on its effectiveness.

IV. DYNAMIC FAULT LOCALIZATION

Dynamic fault localization refers to analyzing error events occurring at runtime in a system to isolate the faulty component. As of this study, we have limited our fault localization to the hardware components in a system (Compute Node1 in Figure 1.) In this section we describe a non-statistical method for dynamic fault localization that uses interaction semantics between components to localize faults. During the design phase of the system, the semantics of the inter-component interactions drive the system architecture. These interactions inherently contain intuitive information about failure symptoms. At design time, based on the hardware components' observable failure states, the failure monitoring mechanisms are arrived at. When the system is in operation, the failure monitoring mechanisms observe for the failure states and report them as error events. How a component's failure affects the other components depends on the interaction semantics of the components, which can be derived based on the functions each component is rendering. By capturing the failure-effect of the interacting components one can localize the faulty component, which in-turn minimizes the dependence on statistical data. This is extremely meaningful for new systems.

Our approach to fault localization is based on analyzing the errors events generated at runtime. This is carried out in four steps as detailed below:

1. Error Event Generation
2. Spatial Classification
3. Temporal Grouping
4. Root Cause Inference.

Error Event Generation: Whenever an erroneous condition, occurs in the hardware, certain error registers of that hardware are set[1]. These registers are monitored by firmware or operating system software components called hardware event monitors. These event monitors are normally programmed to poll periodically for the error conditions or are signaled to wake up on specified errors [17]. Hardware faults in a system are detected dynamically by these monitors and are registered as events, which are then reported by the operating system in the system logs. An incoming error event is extracted from the system log and is parsed to extract the information necessary for deriving spatial and temporal segregation as described later. For the rest of the paper, we will use the term system logs or event logs interchangeably to mean logs containing hardware error events reported at runtime.

Spatial Classification: Most enterprise class systems are architected in a way that hardware error propagation in a component is contained using Error Correcting Codes and parity error checks [18]-[19]. As a consequence, most of the effects of failing component are designed to be visible within a spatial extent. For this paper, we term such a spatial extent (within which errors are contained) as the sub-system. For example, any disk adapter error would be restricted to the IO sub-system of which the adapter is a part, and not propagate to the processor sub-system. In this study, identifying each event with respect to the sub-system it originated from does spatial classification of events. Spatial classification is necessary since the system's log may contain multiple sub-system error events reported in temporal proximity, due to multiple hardware errors occurring almost at the same time. As an error event arrives, it is first sorted into the appropriate spatial group. Events in each spatial group are the separated into temporal groups as detailed below.

Temporal Grouping: Within a sub-system, for a permanent hardware error, the error set, containing the associated error events, cycles repeatedly till some action is taken to rectify the hardware error. This repetition, in the worst case, is observed once every polling interval. For performing fault localization,

[1] This is true in the case of all enterprise class servers.

we need to isolate such error sets. For identifying the repetitive error sets we define the term "interval of influence" (IoI). IoI is used in this paper to denote the time within which a related group of events are logged. The relation of events is with respect to the influence a hardware error has in the generation of that event. Since error events are noticed once in a polling interval, in this study, the IoI was set equal to the polling interval of the associated monitors of the sub-system chosen. From the spatial group, using the IoI, temporal event groups are formed. The temporal groups form the basis for error inference.

Root Cause Inference: For each sub-system, we derive inter-component failure-effect relationship using the component interaction semantics and capture them as Bayesian Belief Networks (BBNs) [20]-[21]. BBNs model problems that involve uncertainty. A BBN is a directed, acyclic graph, whose nodes are the uncertain variables and whose edges represent the causal relation between the variables. In a BBN, each node is assigned certain states with prior probability, i.e. independent probability of the state's occurrence. The cause-effect relations are captured as inter-node state associations with a conditional probability of occurrence, i.e. one state influencing the occurrence of the other. The BBNs use Bayes' theorem for deducing the probability of occurrence of a certain state in a node given the prior probability of the state and the conditional probabilities of the affecting states.

By studying all possible events that can be monitored in a component, we find that there are basically two categories, one that are direct reflection of failures in the component and the other that are symptoms of external failures. For example, in an IO-subsystem, a failure in a disk is normally preceded with timeout and congestion errors on the switch or adaptor connecting to it. These timeout and congestion errors are the symptoms and not direct reflections of failure. The uncertainty here is that the symptoms may not be observed on the failing component. Usually, the symptoms appear before the cause and by capturing the cause-effect relation between the symptoms and failure cause, we can notice the failures ahead of their occurrence. BBN models provide an intuitive way to model these cause-effect relationships and also provide a good mechanism to quantify this relationship. In our model, as depicted in Figure 2, each node in the BBN is either the component that can fail or the monitor that is observing the error event being generated at runtime. A component is represented by two states, Working or Failed and the monitor is represented by the states corresponding to observable events. The cause-effect relation between the observed events and the affected component are captured as the conditional probability values.

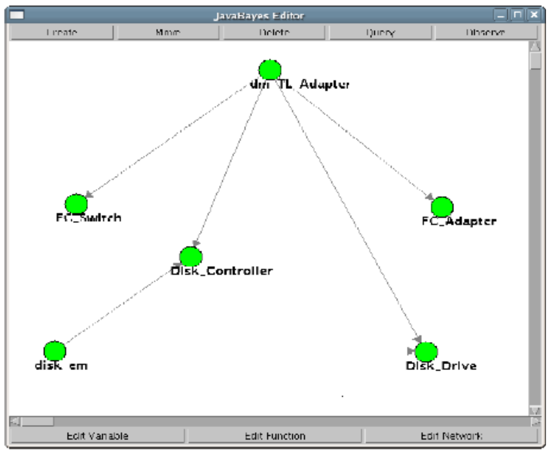

Figure 2: Depicts the Bayesian Belief Network Model for the Fibre Channel Storage Subsystem used in the case study for fault localization. The picture denotes only a subset of the components and monitors of the sub-system. This model was generated using JavaBayes software available at [23].

These conditional probability values quantify the belief of dependence among the related nodes and their corresponding states. For this study these values are derived from the system architecture, based on the interaction semantics of the components. Within BBNs independent nodes are designated prior probability values representing their independent chance to be in Working/Failure states. We have derived the prior probability values of the components from the MTBF values that are vendor specified. Although the component MTBF values are used to assign prior failure probability, it is the interaction semantics between the components that decide the runtime component failure probabilities, based on the error events observed. For building the sub-system BBN model, we followed the method described in [22]. The fault localization process is bootstrapped with a Bayesian Belief network for each sub-system. For each sub-system, events in a Temporal Group are marked as evidence of underlying error and an inference is drawn for failure probabilities of the components in the sub-system. Every time an inference is drawn, the current component availability information is derived from the failure probabilities and projected as the health vector of the system hardware.

V. Availability Prognosis

Using the current availability information we make availability prognosis for the component. The basis for this prognosis is as follows: a component fault is noticed once within the polling interval of the monitor and the resulting error is used to yield the failure probability of the localized component. Until the component fault is rectified or a new component replaces the faulty one, the error event will be repeated periodically and, in the worst case, noticed every polling interval. We use this information to define the decay rate of the component.

For the subsystem we have modeled the polling interval is 1 hour and by observing the errors once in a polling interval, we arrive at the decay rate for the sub-system components to be equal to 1. The expected failure probability is calculated using an exponential decay function. The drop in availability is then calculated as *1-P(F),* where *P(F)* represents probability of failure after time *t*. A(t) is the predicted availability at time *t* and A_c is the current availability. Following equations show these calculations:

$P(F) = e^{-\alpha t}$, and
$A(t) = A_c(1-e^{-\alpha t})$, $e^{-\alpha t}$ is the component decay function,

α is the decay rate and set to 1 for our study,
and t is time in future for which we want to calculate the expected component availability.

VI. Case-Study for dynamic fault localization

In order to assess the correctness of the fault localization method, we applied this method to analyze hardware error event logs generated on a system that was experiencing disk media errors in its IO sub-system. The IO sub-system consisted of a Fibre Channel Adapter connecting to a Fibre Channel Switch, which connected to the failing disk device. The error event data collected was for a period of 2 years from a HP customer system. A HP field engineer independently analyzed this data to identify the faulty disk media. The spatial and temporal grouping algorithm segregated error events into groups that initially had timeout and congestion errors on the adapter followed by intermittent disk controller errors (Figure 3). Subsequently, after a period of 14 hours disk media errors were reported. The BBN for this subsystem reported 75% failure probability with the initial event groups and subsequently the media error pinpointed to a disk failure; Figure 4 depicts the scenario.

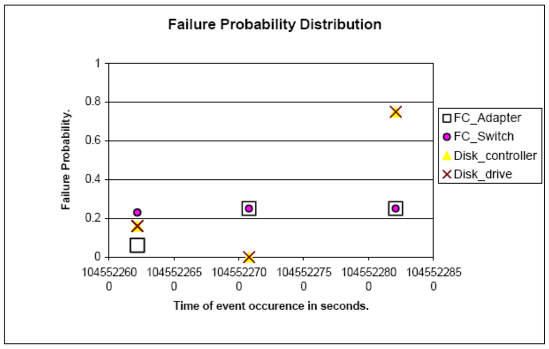

Figure 3: Failure Probability of IO subsystem components inferred with timeout and congestion error events followed by intermittent disk controller error.

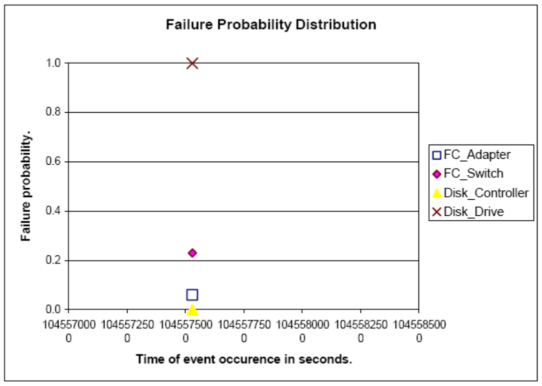

Figure 4: Failure Probability of the components inferred with disk media error events after a period of 14hrs.

Using the observed component failure probabilities, we make an availability prognosis for a period of two hours from the observed values and compare with the actual observed failure probabilities two hours later. Figure 5 and **Error! Reference source not found.** show the trend curves for the current and predicted availability for the disk and disk controller as observed from the event logs of the system. We make an optimistic prediction in less than 10% of the cases. This is shown by the divergence of the predicted trend curve from the current trend curve. Observing the error event data in these cases indicates that the deviation of observed failure probability from the predicted value is due to the occurrence of an error event that helps in pinpointing the cause. To illustrate, when the timeout and congestion errors occur, the BBN indicated equal chances of failures on all of the IO sub-system components. This changes to increased disk failure probability due to the intermittent disk controller error (Figure 3). At this point the predicted availability was 0.82 whereas the observed availability is 0.25.

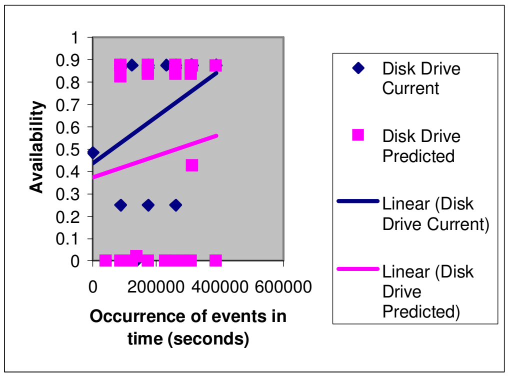

Figure 5: Trend curves for disk drive current and predicted availability.

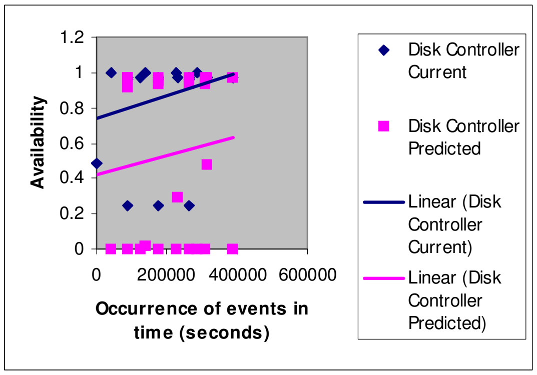

Figure 6: Trend curves for disk controller current and predicted availability.

This observation leads us to conclude two things:

1. An application that got started due to acceptable predicted resource availability could continue till the current resource availability reaches an unacceptable threshold, wherein it can start a recovery procedure.
2. This is true in cases where the failure symptoms are visible well ahead in time before the actual failure. The initial failure symptoms (that of timeout and congestion) in our case study coincide with conditions that can occur intermittently due to performance bottlenecks. Hence, by noticing a small drop (25%) in the current resource availability we cannot conclude a failure possibility as indicated by the prognosis, but a significant drop (75%) definitely indicates a problem.

VII. Enhancements and further research

We propose the following enhancements to this method:

1. The case study described was used for proof-of-concept. The BBN was built using a limited set of observed error events. However, to study utility of this method on a production machine we need to extend the BBN to contain all observable error events. These events may be a few tens of events. Analyzing event relationships with respect to interaction semantics of the components may become intricate and complex. We plan to formally model the sub-system interaction semantics with a view to automatically generate the sub-system BBN. Since the conditional probability values used for inference within the BBN is derived from the system architecture, we plan to conduct sensitivity tests on the BBNs to establish lower and upper bounds on the conditional probability values.
2. It is possible that the BBNs generated could have missed interactions. While testing the BBNs we plan to evaluate them using actual data from many running systems and also integrate with a learning system to arrive at acceptable conditional probability values.
3. As of this study we have evaluated the dynamic hardware fault localization method. We plan to evolve a dynamic method for analyzing and localizing the effect of system software errors and use the combined effect of hardware and system software to arrive at the process health vector.
4. We also plan to integrate the process health vector with AGMETS. At the time of AGMETS development, the testing was done under simulated conditions.

VIII. Conclusion

In this paper we bring out the issues of failures while running applications over distributed resources. We propose a proactive fault-tolerant framework to bring out the context information needed to enable pro-active management. We state the need and a novel method to represent resource availability information and show how the framework uses this information, to provide failure transparency to the application, in face of failures. We also describe a method to quantify the resource availability information at runtime. Using a case study we bring out the effectiveness of this method.

References

[1] Ian Foster, C. Kesselman, S. Tuecke, Anatomy of the Grid: Enabling scalable virtual organizations, International Journal of Supercomputer Applications, 15(3), 2001.
[2] Ian Foster, C. Kesselman, Jeffery M. Nick, S. Tuecke, Grid Services for Distributed System Integration, IEEE Computer, June 2002, p37–46.
[3] Foster, Globus Toolkit Version 4: Software for Service-Oriented Systems. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005.
[4] Steven Newhouse, David Berry, Malcolm Atkinson, Savas P, “Service Grids: Current Activity and Requirements”, [online] available: http://www.nesc.ac.uk/talks/415/intro.pdf
[5] P. Narasimhan, T. A. Dumitras, A. M. Paulos, S. M. Pertet, C. F. Reverte, J. G. Slember and D. Srivastava, MEAD: Support for Real-Time Fault-Tolerant CORBA, Concurrency and Computation: Practice and Experience, vol. 17, no. 12, 2005, pp. 1527-1545; Copyright 2005 John Wiley and Sons.
[6] Mohammad Tanvir Huda, Heinz W. Schmidt and Ian D. Peake, “An Agent Oriented ProActive Fault-tolerant Framework for Grid Computing”, Proceedings of the First International Conference on e-Science and Grid Computing (e-Science’05), IEEE, Dec. 2005.
[7] Ting-Ting Y. Lin, Daniel P. Siewiorek, Error Log analysis: Statistical Modeling and Hueristic Trend Analysis, IEEE Trans. On Reliability, Vol. 39, No. 4, p 419-432, October 1990.

[8] Inhwan Lee, Ravishankar K. Iyer, and Dong Tang, Error/Failure Analysis Using Event Logs from Fault Tolerant Systems, FTCS-21, Computing Digest of Papers, p10-17, June 1991.

[9] Michael F. Buckley' and Daniel P. Siewiorek, VAX/VMS Event Monitoring and Analysis, FTCS-25, Computing Digest of Papers, p414-423, June 1995.

[10] M.F. Buckley and D. P. Siewiorek, Comparative Analysis of event tupling schemes, FTCS-26, Computing Digest of Papers, p294-303, June 1996.

[11] R. K. Sahoo, A.J. Oliner, I. Rishi, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, A. Sivasubramaniam, Critical Event Prediction for Proactive Management in Largescale Computer Clusters, Proceedings of the 9th ACM SIGKDD, p426-435, August 2003.

[12] Ramendra K. Sahoo, Myung Bae, Ricardo Vilalta, Jose Moreira, Shenga Ma, and Manish Gupta, Providing Persistent and Consistent Resources through Event Log Analysis and Predictions for Large-scale Computing Systems, In SHAMAN, Workshop, ICS'2002, June 2002.

[13] Salfner F, Predicting Failures with Hidden Markov Models, In Proceedings of the fifth European Dependable Computing Conference (EDCC 5),Student Forum, Budapest, April 2005.

[14] Nainwal, KC and Lakshmi, J and Nandy, SK and Narayan, Ranjani and Varadarajan, K (2005) A Framework for QoS Adaptive Grid Meta Scheduling. In Proceedings Sixteenth International Workshop on : Database and Expert Systems Applications, 2005, pages pp. 292-296, Denmark.

[15] Kalash Chandra Nainwal, Adaptive Grid Meta-Scheduling – A QoS Perspective, MSc (Engg). Thesis, SERC, IISC, May 2005.

[16] David Patterson, et. Al., Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies, Computer Science Technical Report UCB//CSD-02-1175, U.C. Berkeley, March 15, 2002

[17] HP's EMS Hardware Monitors User's Guide and Event Descriptions found at http://docs.hp.com.

[18] Overview of HP Integrity rx1620-2, rx2620-2 and rx4640-8 Servers, A technical white paper from HP, available online at http://h71028.www7.hp.com/ERC/downloads/5982-9835EN.pdf

[19] Overview of SunFire 15K/12K – Chapter 3 Reliability, Availability, and Serviceability, available online at http://www.sun.com/products-nsolutions/hardware/docs/html/806-3509-12/3__RAS.html

[20] Marek J. Druzdzel and Linda C. van der Gaag, Building Probabilistic Networks: Where Do the Numbers Come From?, IEEE Transactions on Knowledge and DataEngineering, 12(4); 481-486, 2000.

[21] Finn V. Jensen, Bayesian Networks and Decision Graphs, Springer Publication, August 2001.

[22] Martin Neil, Norman Fenton, and Lars Neilson, Building Large-Scale Bayesian Networks, Knowledge Engineering Review, 15(3) vol. 15, no. 3, pp. 257-284, 2000.

[23] JavaBayes software found at http://www.cs.cmu.edu/~javabayes/Home