Detecting Faults in Computational Grids

Russ Wakefield
CS 451 Project Paper

Introduction

Computational grid computing has emerged as one of the most promising research areas in high performance and distributed computing. The ability to allocate large numbers of resources for a parallel application with a greatly reduced cost makes the concept grid computing highly appealing. The heterogeneous nature of a grid creates a complex application environment; the nature of faults within this environment significantly complicates the life of the developer. Creating an application that can recognize and handle these faults in grids becomes non-trivial and complex.

In this paper we will first present a basic definition and a brief history of grid computing since its inception during the last decade. We will then look at a review of the most common faults occurring within the grid environment as identified by a survey of grid computing users. Two papers addressing fault detection are then reviewed for comparison. The first was presented in 1999 from the Globus consortium - a team dedicated to creating open source grid computing software. The paper details their design and implementation of a failure detection component of the Globus toolkit. The second paper was presented in 2005 and is from an independent team of researchers detailing their failure detection model. The focus of their paper is establishing a peer to peer communication system for the purposes of failure detection and information distribution.

Overview

The term grid computing was coined in the late 1990s to describe a set of resources distributed over wide-area networks that can support large-scale distributed applications [1]. The similarities between a vision of readily available computer resources and the existing power grid boosted the analogy – the desire to be able to "plug" into the computational grid and have access to computation and data is likened to plugging an electrical device into an outlet and drawing power to operate it. The lure of these being able to utilize these easily accessible resources has provided the focus for a tremendous amount of research and development. As more and more applications begin using the grid environments, data about the running the applications within that environment becomes available.

A keystone paper defining issues associated with the grid was "An Anatomy of the Grid"[2] by Foster, Kesselman, and Tuecke. "An Anatomy of the Grid" added the concept of a virtual organization to the definition of grid computing – defining the VO as a set of participants with various relationships that wish to share resources to perform a task. Resources are defined to include data, computers, scientific instruments, software, etc. The coordination of these resources provide several key issues, such as complete

sets of resources may be necessary to acquire at once to accomplish a task and the available resources may be changing as participation in the virtual organization is dynamic as participants may join or leave at any time. Other issues include the requirement for flexible resource sharing agreements and clear definition of who the consumers / producers are for management purposes.

Common causes of faults

The fact that the membership of these virtual organizations is made up of diverse types of resources presents its own set of challenges. Distribution, complexity, high heterogeneity, and multiple administrative domains offer technical challenges to the success of using the grid successfully. There are potentially thousands of resources, services, and applications that need to interact in order to use the grid effectively, with all these resources the opportunity for failure is greatly increased. These failures can be caused by independent failures at the element level including hardware/ software/network or due to issues arising from interactions between the elements(such as the interaction between incompatible software components).

According to a survey of users of parallel supercomputers conducted by Medeiros, Cirne, Brasileiro, and Suave[3], the most common types of problems are related to the environment configuration – almost 76% of the responses indicated this as a key issue. Given the nature of the virtual organizations, the lack of control over the grid resources is to be expected – overcoming this lack of control relates back to the need for clear agreements for resource sharing. Middleware failures, application failures, and hardware failures were the top remaining issues within the grid environment according to the survey.

Once the failures had occurred, the fault identification mechanisms were surveyed. Of course the first and most apparent ways were ad hoc mechanisms such as log file analysis and user complaints. Of the automated mechanisms, 57% of the responses had the faults being identified by application-specific mechanisms. Interestingly enough, only 29% of the responses were identified by monitoring software. The other top two responses for fault identification were checkpointing/recovery resulting from crashes and fault-tolerant scheduling – both after the fact.

The survey had several other factors – but the other main key question regarding failure detection mechanisms was the one identifying which issues associated with recovering from a failure. The top issue identified was ability to diagnose the failure required the user to be heavily involved – this was indicated by over 71% of the responses. Almost half of the responses identified difficulty to implement the failure recovery behavior as a problem, with gaining authorization to correct the faulty component listed by 14% as well.

The authors of the survey contend that most of the solutions available for grid fault treatment were designed with performance analysis in mind. These solutions [4] [5] [6] [7] are more focused on returning information and grid monitoring than fault detection

and identification – although the information gathered by these solutions definitely has value in defining a more generic solution.

Let us examine two papers addressing specific solutions to identifying faults within the grid environment.

Globus toolkit

The first paper "A Fault Detection Service for Wide Area Distributed Computations" from Stelling, Foster, Kesselman, Lee, and Laszewski [8] is a product of the development work of the Globus project. The Globus consortium identifies itself as a non-profit organization formed by global computing leaders who support the Globus Toolkit, the de facto standard for open source grid computing infrastructure [9]. The Globus Toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability [10]. This paper describes a fault detection service based on unreliable fault detectors and its implementation as the Globus Heartbeat Monitor. Unreliable fault detectors are defined as detectors that will sometimes report that a component has failed and then retract that report at a later time.

The authors contend that unreliable fault detectors can be distributed with each component of the application system having access to its own detector. These detectors can potentially produce a different account of which components have failed – as long as all failed components are eventually discovered and identified as such and at least one functioning component is known to be functioning by all functioning components in the system after some point in time. Using this methodology in a consensus algorithm where all the components have to agree that a failed node is failed provides the minimum functionality required for the service.

Unreliable failure detectors also have several advantages over reliable failure detectors. Allowing each component to have access to its own failure detector without requiring the detectors to agree which components of a system have failed keeps the failure detection service from being centralized. Unreliable failure detectors can also used unreliable communication protocols for the same reason. Unreliable communication protocols have lower overheads, reduced latency, and provide better scalability.

The downside to unreliable failure detectors is the requirement that *eventually* the detector will identify all failed components and at least one functioning component. This wait can be unacceptable if the cost of waiting for the correct determination exceeds the cost of simply accepting the failure as fact and resolving the issue as if the failure was fact. Whether this is important or not is a function of the application – it must take the information provided by the detector and make a decision as to the action to take based on the probability the information is correct.

The key point the authors make about unreliable failure detectors is that a distributed computing environment should provide unreliable failure detectors as a basic service – providing information about the possible failure of components is the environment's

responsibility and it is up to the application to determine the reliability of the information and take the appropriate action. Since this is part of their basic toolkit for distributed environments – you can understand their view.

The model defined by the authors includes a local monitor – responsible for monitoring the computer on which it runs as well as selected processes on that computer, a client registration API – which an application can use to identify which processes to be monitored by the local monitor, and a data collector API – allowing an application to be notified about events pertaining to the monitored process. An application that wants to use the author's service registers the processes for which failure detection is required and uses the data collector API to build an application-specific collector to analyze the data and take action based on that work.

Cluster communication protocol

The second paper, "A Scalable and Efficient Self Organizing Failure Detector for Grid Applications"[11] by Horita, Taura, and Chikayama was written over 5 years after the first paper. In that time, significant work had occurred in failure detection for grid computing – their focus assumed some of the points of the first paper, and brought a more narrow focus to the problem. The authors proposed a system in which each node is monitored by a randomly chosen small set of peer nodes, that each node is monitored by TCP connections and heartbeats on them – a process crash is detected by a connection reset and a machine crash detected by the absence of heartbeats, and that once faults are detected, notifications are quickly disseminated along the established TCP connections by simple flooding. This allows the proposed system to work with almost no manual configuration about network connectivity (such as firewalls) – and without a monitoring process on each system. They proposed this functionality would be contained in the grid application libraries such as PVM (Parallel Virtual Machine).

Like the first paper, the author's proposed system detected failures by using a heartbeat system – the absence of heartbeats detects the failures. Since operating systems of today close TCP connections when a process is terminated – the heartbeat is no longer seen by processes that have established a connection. Using heartbeats has known issues – detection latencies must be long enough to reduce false positives, heartbeats can go missing for a variety of reasons that have nothing to do with faults, and even if you do see a heartbeat, it doesn't necessarily imply the process has not had a fault. The authors saw these issues less important than the ease achieved by using the heartbeat mechanism – both OS/hardware faults and process deaths are detected using heartbeat. Like the first paper – no effort was made to detect Byzantine behavior in the application – this is non-trivial to solve, and very application-specific.

The authors defined the following as desirable traits in a failure detector:

- Completeness – any failure is eventually detected by all normal processes
- Accuracy – the probability of false positives is low

- Consistency – all processes that are not declared as failed receive consistent information – including false positives
- Detection Latency – latency between failure and detection low
- Scalability – System load kept to a minimum
- Flexibility – various network settings (firewalls) are tolerated
- Adaptiveness – must be accomplished without a lot of manual configuration

The first five are defined as standard properties to be desired in a failure detection mechanism, the list matches many of the traits defined in the earlier paper. The last two properties (flexibility and adaptiveness) are the key two points the authors wish to have added to the list to support their proposal.

The proposed system (protocol) breaks the problem into two phases – failure detection and information distribution. Their protocol establishes that each process is monitored by some number – defined by a configurable constant – of peer processes. This constant is small, typically four or five. Once one of these monitoring processes detects a failure, the information is propagated to all the normal processes – normal being defined as a process not currently in failure mode. The goal of the failure detection phase is to establish and maintain the monitoring connections between the peer processes. The information propagation phase involves flooding on all the established connections notification of the failure detection. Since each node knows about the connections it has established to monitor itself as well as the connections it has accepted to monitor others, the total traffic is 2 * (the constant defining the number of monitoring processes) * (total number of processes). The authors had not incorporated any consensus algorithm to reduce the frequency of false positives.

When extending this concept to a wide area network, the randomness of the connectivity graph becomes essential. If you allow the processes to choose their own monitors, it is likely the processes would choose nodes within their own cluster and isolation of the clusters becomes possible. Gateway nodes into clusters are likely not to be chosen and the authors had defined adaptiveness – little manual configuration – as a key property desirable in a failure detector. The authors then presented an algorithm as the key info of the paper to resolve this issue. This algorithm is based on readily available information allowing the no manual configuration property to be achieved.

Within the cluster, a process behaves as defined above. If the number of connections between two clusters is less than the monitoring constant defined above, when a new process connects within a cluster, it tries to connect to a process within the other cluster as one of its monitoring processes. Once the connection is established, notification of the connection is broadcast to the members of the two involved clusters. This allows the members to keep an approximate count. If the new count is greater than or equal to the monitoring constant, the process stops attempting cross-cluster connections. To resolve the definition of a cluster – it is all nodes that share the same subnet mask. Therefore, two processes can determine if they belong to the same cluster by examining the subnet mask contained in each message.

Summary

The desire to use available resources to solve large scale problems has provided a challenge to researchers to find solutions to make this model work. Grid computing – by the nature of wanting to use those available resources – offers many challenges to work within the distributed environment. These challenges include heterogeneous equipment, differing administrative domains, and complexity. Discovering and resolving faults is both critical to the success of working within the grid environment and problematic due to its nature.

The main issue associated with fault identification is a cognitive one. The nature of the grid makes understanding all the components associated a difficult one, not unlike many of the problems solved by computer scientists over the years. It will be necessary to create layers of abstraction to allow us to narrow the scope of these faults to be able to identify solutions. The two papers presenting fault detection solutions illustrate the fact that this work is on-going. In the 1999 paper, the authors are approaching the problem from a framework view, establishing the necessary pieces to get a minimum successful architecture necessary to provide the service as a part of the environment. In the 2005 paper, the authors are taking the work done by the first crew (although in a different framework) as a given and are offering refinements to the protocols to improve the communication paths within the distributed system.

A survey was presented offering several views into the users perceptions of the issues they are currently having using the grid environments. The data presented seems fairly intuitive – the number one issue viewed as causing faults is one of configuration management. Again – as the frameworks continue to be developed (this survey was presented in 2003), these issues will be helped by the layers of abstraction being created as a part of the framework.

The first paper was a design and description of a fault detection system that needed to be created as a part of an overall framework, and as such was focused on the activities and utilities necessary to make that a success. Since this is now a part of the Globus toolkit – their website continues to provide information with respect to the failure detection services – clearly this a model that is in production and working. There were no papers available describing refinements to the service. The second paper was more esoteric – a description of a model that had prototypes built but was not incorporated into the PVM architecture the authors referenced in their paper. My concern of their approach is one of scalability. The point to point connection probably works well within small numbers of clusters holding large numbers of nodes – but if the clustering is scattered (i.e. many clusters holding few nodes) – the ability to create point to point solutions within each cluster seems problematic.

References

[1] H. Cassanova. Distributed Computing Research Issues in Grid Computing. ACM SIGACT News Distributed Computing Column (8), 2002.

[2] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Allications, 15(3), 2001.

[3] R. Medeiros, W. Cirne, F. Brasileiro, J. Sauvé. Faults in Grids: Why are they so bad and What can be done about it? Proceedings of the IEEE Fourth International Workshop on Grid Computing, 2003.

[4] B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski, and M. Swany. A grid Monitoring Architecture. Working Document, January 2002. http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-2.pdf

[5] W. Smith. A Framework for Control and Observation in Distributed Environments. NASA Advanced Computing Division, NASA Ames Research Center, Moffett Field, CA, NAS-01-006, June 2001.

[6] B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee, and M. Thompson. A Monitoring Sensor Management System for Grid Environments. Proceedings of the IEEE High Performance Distributed Computing Conference (HPDC-9), August, 2000.

[7] A. Waeed, W. Smith, J. George, and J. Yan. An Infrastructure for Monitoring and Management in Computational Grids. In Proceedings of the 2000 Conference on Languages, Compilers, and Runtime Systems, 2000.

[8] P. Stelling, I. Foster, C. Kesselman, C. Lee, G. von Laszewski. A Fault Detection Service for Wide Area Distributed Computations. Proceedings of the 7th IEEE Symposium on High Performance Distributed Computing, 1998.

[9] http://www.globusconsortium.org/

[10] http://www.globus.org/toolkit/about.html

[11] Y. Horita, K. Taura, T. Chikayama. A Scalable and Effiecient Self Organizing Failure Detector for Grid Applications. Proceedings of the 2005 IEEE Grid Computing Workshop, 2005.