REAL-TIME TRANSACTION PROCESSING FOR AUTONOMIC GRID APPLICATION

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology

In

Computer Science and Engineering

By

BANALATA SARANGI

Department of Computer Science and Engineering

National Institute of Technology

Rourkela

2007

REAL-TIME TRANSACTION PROCESSING FOR AUTONOMIC GRID APPLICATION

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology

In

Computer Science and Engineering

By

BANALATA SARANGI

Under the guidance of

BIBHUDATTA SAHOO

Department of Computer Science and Engineering

National Institute of Technology

Rourkela

2007

National Institute of Technology

Rourkela

CIRTIFICATE

This is to certify that the thesis entitled “Real-time Transaction Processing for Autonomic Grid Application” Submitted by Ms Banalata Sarangi in partial fulfillment of the requirements for the award of master of Technology degree in computer science and engineering with specialization in “Computer Science” at the National Institute of Technology, Rourkela (Deemed University) is an Authentic work carried out by her under my Supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other university/Institution for the award of any Degree or Diploma.

Bibhudatta Sahoo

Senior Lecturer

Department of CSE

National Institute of Technology

Rourkela-769008

Date 21 may 2007

ACKNOWLEDGEMENTS

No thesis has been created entirely by an individual; many people have help to create This thesis and each of their contribution has been valuable. I express my sincere gratitude to my thesis supervisor, Prof Bibhudatta Sahoo for his kind and valuable guidance for the completion of this thesis work, His consistent support and intellectual made me energize and innovate new ideas. I am grateful to Dr.S.K.Jena, Professor and Head, CSE for his excellent support during my work. I am also thankful to Dr. S.K.Ratha, Dr.B.Majhi, Dr.D.P Mohapatra, and Dr.A.K. Turuk, and Dr.R.Baliarsingh of CSE department for providing me support and advice in preparing my thesis. Thanks to all my classmates for their love and support. Last, but not the least I would like to thanks my parents and family member for supporting me to complete my master degree in all ways.

Banalata Sarangi

Contents

Abstract

The advances in computing and communication technologies and software have resulted in an explosive growth in computing systems and applications that impact all aspects of our life. Computing systems are expected to be effective and serve useful purpose when they are first introduced and continue to be useful as condition changes. With increase in complexity of systems and applications, their development, configuration, and management challenges are beyond the capabilities of existing tools and methodologies. So the system becomes unmanageable and insecure. So in order to make the systems self-manageable and secure the concept of Autonomic computing is evolved. Autonomic computing offers a potential solution to these challenging research problems. It is inspired by nature and biological systems (such as the autonomic nervous system) that have evolved to cope with the challenges of scale, complexity, heterogeneity and unpredictability by being decentralized, context aware, adaptive and resilient. This new era of computing is driven by the convergence of biological and digital computing systems and is characterized by being self-defining, self-configuring, self-optimizing, self-protecting, self-healing, context aware and anticipatory. Autonomic computing is a new computing model to self manages computing systems with a minimal human interference. It provides an unprecedented level of self-regulation and hides complexity from Users. The Autonomic computing initiative is inspired by the human body's autonomic nervous system. The autonomic nervous system monitors the heart- beats, checks blood sugar levels and maintains normal body temperature with out any conscious effort from the human. There is an important distinction between autonomic activity in the human body and autonomic responses in computer systems. Many of the decision made autonomic elements in computer systems make decisions based on tasks, which are chosen to be delegated to the technology. The influences of the autonomic nervous

systems may imply that the autonomic computing initiative is concerned only with low-level self-managing capability such as reflex reaction. The basic application area of autonomic computing is grid computing. Both autonomic computing and grid computing are proposed as innovations of IT. Autonomic computing aims to present a solution to the rapidly increasing complexity crises in IT industry, as grid computing tries to share and integrate distributed computational resources and data resources. Basic aim is to implement the autonomic computing in grid related study like autonomic task distribution and handling in grids, and autonomic resource allocation.

In this thesis paper we presents methods of calculating deadlines of global and local transaction And sub transaction by taking EDF algorithm and measure the performance by taking miss ratio in Different workload. We implement this work in an existing grid.

The basic aim is to know autonomic computing better. It is a model to self manage computing Systems with minimal human interference. Self manage has properties like self-configuration, self-optimization, self-healing, self-protection. Autonomic grid computing combines autonomic computing with grid technologies to help companies to reduce the complexity associated with the grid system and hides the complexity from their grid user. Autonomic real-time transaction services incorporate fault tolerance into autonomic grid technology by automatically recovering systems from various failures.

Here in this paper Deadlines of global transaction, sub transaction and local transaction are calculated by taking parameters arrival time, execution time, relative deadline, and slack time. We are taking a periodic transaction having λ (transaction arrival rate per second) Tasks are generated at different nodes with Poisson ratio with λ as workload. Miss ratio is the performance metrics. With increase in workload miss ratio first decreased and then rose. The reason was each sub transaction acted as a unit to compete for resources so that more workload the more system resource they consumed. So more transaction missed their deadlines, as they could not get enough resource in time. EDF algorithm has both less global and local miss ratios then other scheduling algorithm. If EDF is compare with FCFS or SJF or HPF it is apparent that both algorithms perform almost identically until no of transaction is low, then EDF misses fewer dead lines than other. Real-time transaction can handled by the grid in autonomic environment and satisfy properties of autonomic computing.

List of figure

CHAPTER-1

INTRODUCTION

Recent years have seen an explosive growth and wide deployment of Internet technologies, the proliferation of networked systems/devices, services and applications, and the emergence of a ubiquitous information Grid. These pervasive computing and communication technologies are rapidly weaving themselves into the fabrics of everyday life and are fundamentally redefining the way we interact with information, each other, and the world around us. However, these systems, services, and applications present significant challenges due to the size, heterogeneity and dynamism of the environment, the volume and diversity of the data, and the lack of quality or integrity assurances. In fact, this growth and proliferation is rapidly leading to unprecedented software/system scales, complexity, heterogeneity, dynamics, unpredictability and unreliability and is threatening to undermine the very benefits that information technology aims to provide. It is rapidly causing existing paradigms, processes and technologies to breakdown and making our computational/information infrastructure brittle, unmanageable and insecure

The increasingly evolving environment of computing and communication systems in conjunction with the need for convergence and integration of existing and future heterogeneous environments introduces a huge complexity in the management of these systems. Such complexity necessitates the introduction of automation in today's and future systems, so as to minimize the need for human intervention.

Autonomic computing is a new computing model to self manages computing systems with a minimal human interference. It provides an unprecedented level of self-regulation and hides complexity from Users. The Autonomic computing initiative is inspired by the human body’s autonomic nervous system. The autonomic nervous system monitors the heart- beats, checks blood sugar levels and maintains normal body temperature with out any conscious effort from the human. There is an important distinction between autonomic activity in the human body and autonomic responses in computer systems. Many of the decision made autonomic elements in computer systems make decisions based on tasks, which are chosen to be delegated to the technology. The influences of the autonomic nervous systems may imply that the autonomic computing initiative is concerned only with low-level self-managing capability such as reflex reaction.

Real-time transaction depends on dead line. Dead line in real time transaction refers to the time by which the transaction must finish or else undesirable result may produce. Depending on the strictness of deal line real time transactions are of three-type mimicking traditional distributed system.

- Hard real-time transaction
- Soft real-time transaction
- Firm real time transaction.

The main goal of the work presented here is to study the behaviors of autonomic computing. This paper presents autonomic real time transaction service (ARTTS), which incorporates fault tolerance in autonomic grid technology by automatically recovering the system from various failures. The ARTTS able to detect and prevent the system wide failures for autonomic grid system and handle the entire process on behalf of user.

Here I present an extensive study on autonomic computing architecture, its working principle, grid as a distributed environment, components of grid architecture, transaction, periodic, a periodic real time transaction and various scheduling algorithm, deadline calculation and finally implementing real-time transaction scheduling algorithms in grid environment to satisfy autonomic properties.

Finally, the application software used is ***MATLAB***, version 7.1 that is a high performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation.

CHAPTER-2

LITERATURE REVIEW

Feilong Tang, Minglu Li, and Joshua Zhexue Huang [1] worked on Real-time transaction Processing for autonomic Grid applications and design and verify the coordination algorithm using petrinet model. They possessed coordination algorithm for the sub transaction and global transaction.

Xiaolong Jin, and Jiming Liu [2] Characterize autonomic task distribution and Handling in grids.

Rainer Unland, and Huaglory Tianfield [3] visualize the aspects of Autonomic computing Systems.

Ricardo M.Bastos, Flavio M.de Oliveira, and Jase Palazzo M.de Oliveira [4] worked on Autonomic computing approach for resource allocation, and provide a model for agent based autonomic resource allocation. Basically they present an autonomic solution based in a multi-agent model for resource allocation in a manufacturing environment.

Steve R.white, James E.Hanson, Ian Whally, David M.chess, and Jeffrey O.Kephart [5] describe An Architectural Approach to Autonomic Computing.

E.Grishikashvili Pereira, R.Pereira, and A.Taleb-Bendiav [6] evaluate the Performance for self-healing Distributed services and fault detection mechanisms.

Andrea Baldini, Alfredo Benso, and Paolo Prinetto [7] discover a dependable autonomic computing environment for self-testing of complex heterogeneous systems.

2.1 Autonomic computing :Implementing the vision

The need for autonomic computing system (technologies that enable systems to be more self managing and require minimal human intervention) is growing everyday as system becomes more complex, more difficult to maintain, and more expensive to manage. IBM has products with autonomic capabilities available today, and is developing new technologies to move customer to the ever-changing levels of autonomic computing.

2.2 Autonomic Computing Systems and Applications

Autonomic applications and systems are composed from autonomic elements, and are Capable of managing their behaviors and their relationships with other systems/ applications in accordance with high-level policies. Autonomic systems/applications Exhibit eight defining characteristics:

Self Awareness**:** An autonomic application/system "knows itself" and is aware of Its state and its behaviors.

Self Configuring**:** An autonomic application/system should be able configure and Reconfigure it under varying and unpredictable conditions.

Self-Optimizing**:** An autonomic application/system should be able to detect sub optimal Behaviors and optimize itself to improve its execution.

Self-Healing**:** An autonomic application/system should be able to detect and recover From potential problems and continue to function smoothly.

Self-Protecting: An autonomic application/system should be capable of detecting And protecting its resources from both internal and external attack and maintaining Overall system security and integrity.

Context Aware: An autonomic application/system should be aware of its execution Environment and be able to react to changes in the environment.

Open**:** An autonomic application/system must function in a heterogeneous world And should be portable across multiple hardware and software architectures. Consequently it must be built on standard and open protocols and interfaces.

Anticipatory**:** An autonomic application/system should be able to anticipate to the Extent possible, its needs and behaviors and those of its context, and be able to Manage it proactively.

2.3 Primary mechanism of autonomic computing

Two mechanism are primary for autonomic computing i.e. system adaptation and complexity hiding from user.

Ac ACS (autonomic computing system) is autonomous in that it completely hides the complexity hiding from users means that autonomic computing will provide users with a computing environment that allows them to concentrate on what they want with out worrying about how it has to be done.

Autonomic computing = system adaptation + complexity hiding
= Automation of system adaptation.

2.4 ACS (autonomic computing system)

The work of an ACS further divided into four groups according to the properties and characteristics as follows.

1. Establishing frameworks of autonomic computing

- Improving Grid service's Qos through self-configuring regulation.
- Autonomic fault migration in embedded systems
- Autonomic networks: engineering and self healing properties
- Autonomic wireless sensor networks.

2. Self-configuration in dynamic software architecture and web security

- Automated adaptations to dynamic software architecture s by using autonomous agents
- A dynamically reconfigurable system based on workflow and service agents.
- Dynamic security reconfiguration for semantic web

3. Automation for autonomic computing

- Real-time transaction processing for autonomic grid applications
- Characterizing autonomic task distribution and handling in grids
- Autonomic resource allocation by multi-agent system.

4. Auto tuning and learning

- Pricing based strategies for autonomic control of web servers for time-varying request arrivals
- Adaptive job routing and scheduling.

2.5 Automation for autonomic computing

The real-time transaction processing is a key and challenging technology to prevent systems in an autonomic Grid environment from various failures. Tang et al. (2004) presents an autonomic real-time transaction service. It can dynamically discover grid services as participants to execute specified sub-transactions, coordinate these participants to achieve the real-time and transactional requirements, and assign priorities to schedule concurrent transactions.petrinet are used to model and validate the co-

ordination algorithms of real-time grid transactions. By handling the potential failures and exceptions in an autonomic manner, the autonomic real time transaction service can facilitate the implementation of real-time transaction and alleviate users from the system administration in Grid environment.

In a grid, numerous tasks need to be distributed in a decentralized fashion. One of the important problems is how to implement task distribution and handling. Jin and Liu present an agent based task distribution and handling paradigm for grid.

CHAPTER-3

AUTONOMIC COMPUTING

3.1 What is autonomic computing?

Autonomic computing is an approach to self-managed computing systems with a minimum of human interference. The term derives from the body's autonomic nervous system, which controls key function without conscious awareness or involvement. Autonomic computing is an emerging area of study and a grand challenge for the entire IT community to address in earnest.

Autonomic computing is a promising new concept in system development. It aims to (i) increase reliability by designing systems to be self-protecting and self-healing; and (ii) increase autonomy and performance by enabling systems to adapt to changing circumstances, using self configuring and self optimizing mechanisms.

Autonomic computing concept is inspired by the human body's autonomic nervous system. Like human have good mechanisms for adapting to changing environments and repairing minor physical damage.

3.1.1 what are the origins of autonomic computing?

Autonomic computing is the evolution of a long tradition of understanding and creating self-regulating systems. It's risen to the top of the I/T agenda because of the immediate need to solve the skills shortage and the rapidly increasing size and complexity of the world'-computing-infrastructure.

3.1.2 what is the goal of autonomic computing?

The goal is to realize the promise of I/T: increasing productivity while minimizing complexity for users. It's time to design and build computing systems capable of running themselves, adjusting to varying circumstances, and preparing their resources to handle most efficiently the workloads we put upon them.

3.1.3 what does autonomic computing promise to deliver?

Most immediately, the automated management of computing systems. But that capability will provide the basis for much more: from seamless e-sourcing and grid computing to dynamic e-business and the ability to translate business decisions that managers make to the I/T processes and policies that make those decisions a reality. Ultimately, autonomic computing is a challenge that must be met before the industry can deliver 'the next big thing.'

3.2 Characteristics of an Autonomic System

- The system must know its own components and their details.
- The system must change its configurations constantly with changing environments.
- The system must continuously look for ways to optimize its process.
- The system must recognize abnormal conditions or problems that may harm its
- Workings and be able to recover from them.
- The system must be protected against attacks.
- The system must know its environment, surroundings, and other resources available to it.
- The system must have open standard and operated in heterogeneous world.
- The system must able to stay ahead of the user and guess intelligently what resources will be required and how to use them efficiently.

3.3 Properties of Autonomic System

An autonomic system is self-managing means it focuses basically on four issues, it is self-protecting, self -configuring, self-healing and self- optimizing.

3.3.1 Self -Configuration

A *self-configuring* system automatically configures themselves according to high-level policies. Self-configuring is a systems ability to readjust itself automatically to changing circumstances. Self-configuration is an important part of autonomic computing vision. Autonomic elements configure themselves, based on the environment they find themselves and the high-level task for which they have been set, with out any detail human intervention in the form of configuration files or installation dialogs. This may simply be in self-healing, self-optimization or self protection.

3.3.2 Self -Protection

A *self-protecting* system will defend it self from accidental or malicious external attack. This means being aware of potential threats and having ways of handling those threats. It must aware of potential threats and having ways of handling those threats. These systems are automatically resist malicious attacks or cascading failures. They use early warnings to anticipate and prevent system-wide failures. There are two distinct but related aspects to self-protection, protecting against undesirable systems behavior due to bugs or unanticipated conditions, and protection against system penetration by attackers.

3.3.3 Self -optimization

Self-optimization means a system is aware of its ideal performance, can measure its current performance against that ideal performance and has strategies for attempting improvements. Components and systems continually seek opportunities to improve their own performance and efficiency.

3.3.4 Self- healing

Autonomic systems automatically detect, diagnose, and repair local software and hardware problems. Self-healing is concerned with ensuring effective recovery when a fault occurs. This means successfully identifying the fault and then, where possible,

repairing it, there should be minimal disruption to users, avoiding loss of data and significant delay in processing.

3.4 Architecture of an autonomic element

Basically we assume that an autonomic computing system is made up of a connected set of autonomic elements. Each element must include a sensor and effectors. Monitoring behaviors is done through the sensors, comparing this with expectation, deciding what action, if any, is needed and then executing that action through the effectors, creates a control loop.

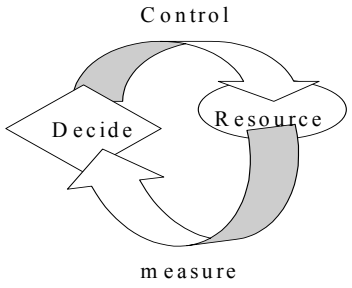


Figure-1 Control loop

Figure-3.1

In each autonomic element there is managed component and a corresponding autonomic manager implementing the required self -monitoring and self- adjusting. An internal monitor observes the state of the managed component and passes this information to the self monitor for evaluation and action. The measured state is compared with the expected state held in a system knowledge base. Undesirable deviations are reported to the self adjuster for action, which may result in changes to the managed component. Similarly an external monitor observes the state of the environment via an autonomic signal channel and this also may trigger internal changes. The signal channel provides linkage to other autonomic manager. The heart beat or pulse monitor provides a summary of the state of an autonomic element to other autonomic elements responsible for monitoring the state.

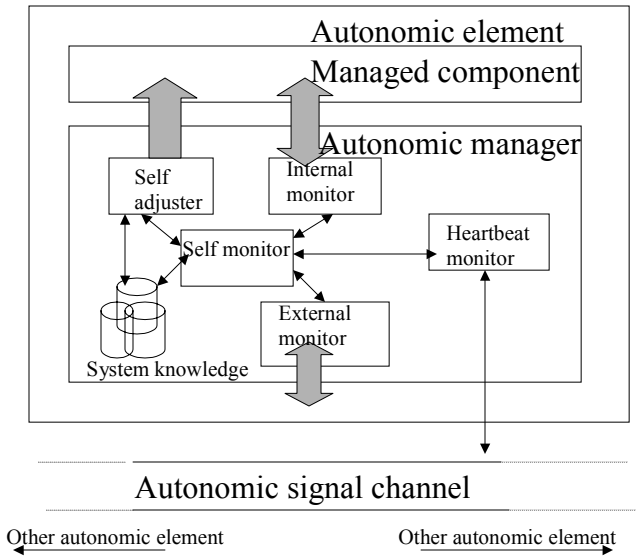


Figure-2 potential architecture of an autonomic element

figure-3.2

3.5 Application of Autonomic computing

The basic application area of autonomic computing is grid computing. Both autonomic computing and grid computing are proposed as innovations of IT. Autonomic computing aims to present a solution to the rapidly increasing complexity crises in IT industry, as grid computing tries to share and integrate distributed computational resources and data resources. Basic aim is to implement the autonomic computing in grid related study like autonomic task distribution and handling in grids, and autonomic resource allocation.

CHAPTER –4

GRID COMPUTING

4.1 Introduction

The popularity of the Internet as well as the availability of powerful computers and high-speed network technologies as low-cost commodity components is changing the way we use computers today. These technology opportunities have led to the possibility of using distributed computers as a single, unified computing resource, leading to what is popularly known as Grid computing. The term Grid is chosen as an analogy to a power Grid that provides consistent, pervasive, dependable, transparent access to electricity irrespective of its source. This new approach to network computing is known by several names, such as Meta computing, scalable computing, global computing, Internet computing, and more recently peer to-peer (P2P) computing. Grids enable the sharing, selection, and aggregation of a wide variety of resources including supercomputers, storage systems, data sources, and specialized devices that are geographically distributed and owned by different organizations for solving large-scale computational and data intensive problems in science, engineering, and commerce.

Thus creating virtual organizations and enterprises as a temporary alliance of enterprises or organizations that come together to share resources and skills, core competencies, or resources in order to better respond to business opportunities or large-scale application processing requirements, and whose cooperation is supported by computer networks.The concept of Grid computing started as a project to link geographically dispersed supercomputers, but now it has grown far beyond its original intent. The Grid infrastructure can benefit many applications, including collaborative engineering, data exploration, high-throughput computing, and distributed supercomputing.

4.2 What is a Grid?

Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed "autonomous" resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements. Grid computing, most simply stated, is distributed computing

taken to the next evolutionary level. The goal is to create the illusion of a simple yet large and powerful self-managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources.

Grid is a collection of distributed resources connected by a network, possibly at different sites and in different organizations. Those resources may include supercomputers, instruments such as telescopes and microscopes, computer-controlled factory floor tools, mid- level servers, desktop machines, laptop etc.

The resources in a Grid typically share at least some of the following characteristics:

- They are numerous.
- They are owned and managed by different, potentially mutually distrustful organizations and individuals that likely have different security policies and practices.
- The resources are potentially faulty.
- They have different security requirements and policies.
- They are heterogeneous, i.e., they have different CPU architectures, are running different operating systems, and have different amounts of memory and disk.
- Heterogeneous, multilevel networks connect them.
- They have different resource management policies.
- They are likely to be geographically separated (on a campus, in an enterprise, on a continent).

4.3 Grid computing

A grid usually connects huge number of computers over the Internet as a complex computational System. Here numerous tasks are distributed to grid nodes in decentralized fashion. It contains five Components.

1. A portal
2. A service broker
3. Task scheduler

4. A task manager
5. A group of grid node.

The portal acts as a user interface, through which user can log in and use the grid. After having logged into the grid, a user can submit a task.

The service broker will check whether or not the grid possesses resources suitable for handling the submitted task. If so, it will further check whether or not the resources are available now.

The task scheduler is responsible for scheduling submitted tasks to be served. The task manager finally launches a submitted task.

The nodes, as the core of a grid, are prerequisites. Grid nodes can be desktops, workstations, and clusters that belong to different LANs, WANs, or the Internet.

4.4 Grid Architecture

In grid computing, many thousands of small-distributed computing networks would be linked over worldwide grids in a Web like system resembling a public utility's power grid. That will let businesses send data transfer, share software more easily and store even more information than today's computer networks. The development of Grid-enabled applications presents a significant challenge, however, because of the high degree of heterogeneity and dynamic behavior in architecture, mechanisms, and performance encountered in Grid environment.

The Grid is made up of a number of components from enabling resources to end user applications. A layered architecture of the Grid and its components are shown in the following Figure.

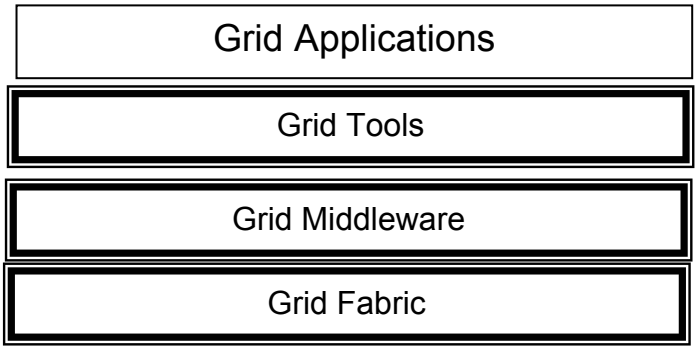


Figure-4.1 Block Diagram of Computational Grid

Grid computing technologies enable controlled resource sharing in distributed communities and the coordinated use of those shared resources as community members tackle common goals. These technologies include new protocols, services, and APIs for secure resource access, resource management, fault detection, communication, and so forth that in term enable new application concepts such as virtual data, smart instruments.

4.5 Grid Components

A computational grid can be modeled using 4-layer architecture as (1) Grid fabric, (2) Core Grid Middleware, (3) Grid Tools, and (4) Grid Fabric.

A Grid Fabric is consists of all the globally distributed resources that are accessible from anywhere on the Internet. These resources could be computers (such as PCs, SMPs, clusters) running a variety of operating systems (such as UNIX or Windows) as well as resource management systems such as LSF (Load Sharing Facility), Condor, PBS (Portable Batch System) or SGE (Sun Grid Engine), storage devices, databases, and special scientific instruments such as a radio telescope or particular heat sensor.

The Core Grid Middleware offers core services such as remote process management, co-allocation of resources, storage access, information registration and discovery, security and aspects of Quality of Service (QOS) such as resource reservation and trading.

The Grid Tools (User-Level Grid Middleware) includes application development environments, programming tools, and resource brokers for managing resources and scheduling application tasks for execution on global resources.

The Grid Applications consists of Grid applications or portals. Grid applications are typically developed using Grid-enabled languages and utilities such as MPI (message-passing interface) or Nimrod parameter specification language. An example application, such as parameter simulation or grand-challenge problem would require computational powers, access to remote data sets, and may need to interact with scientific instruments. Grid portals offer Web-enabled application services, where the users can submit and collect results for their jobs on remote resources through the Web.

4.5.1 Virtual Organizations

Grid *architecture* identifies fundamental system components, specifies the purpose and function of these components, and indicates how these components interact with one another.

Within internal enterprise IT infrastructures, SP-enhanced IT infrastructures, and multi-organizational Grids, computing is increasingly concerned with the creation, management, and application of dynamic ensembles of resources and services (and people)—what is called as *virtual organizations*. Depending on context, these ensembles can be small or large, short-lived or long-lived, single institutional or multi-institutional, and homogeneous or heterogeneous. Individual ensembles may be structured hierarchically from smaller systems and may overlap in membership. Regardless of these differences, developers of applications for VOs face common requirements as they seek to deliver QoS—whether measured in terms of common security semantics, distributed workflow and resource management, coordinated fail-over, problem determination

services, or other metrics—across a collection of resources with heterogeneous and often dynamic characteristics. Nature of these requirements and the mechanisms required to address them in practical settings introduces an Open Grid Services Architecture that supports the creation, maintenance, and application of ensembles of services maintained by VOs.

An extensible and open *Grid architecture*, in which protocols, services, application programming interfaces, and software development kits are categorized according to their roles in enabling resource sharing is described below.

Fabric: Interfaces to Local Control

The Grid *Fabric* layer provides the resources to which shared access is mediated by Grid protocols: for example, computational resources, storage systems, catalogs, network resources, and sensors. A "resource" may be a logical entity, such as a distributed file system, computer cluster, or distributed computer pool; in such cases, a resource implementation may involve internal protocols (e.g., the NFS storage access protocol or a cluster resource management system's process management protocol), but these are not the concern of Grid architecture.

Fabric components implement the local, resource-specific operations that occur on specific resources (whether physical or logical) as a result of sharing operations at higher levels. There is thus a tight and subtle interdependence between the functions implemented at the Fabric level, on the one hand, and the sharing operations supported, on the other. Richer Fabric functionality enables more sophisticated sharing operations.

It is suggested that at a minimum, resources should implement *enquiry* mechanisms that permit discovery of their structure, state, and capabilities (e.g., whether they support advance reservation) on the one hand, and *resource management* mechanisms that provide some control of delivered quality of service, on the other. The following brief and partial list provides a resource specific characterization of capabilities.

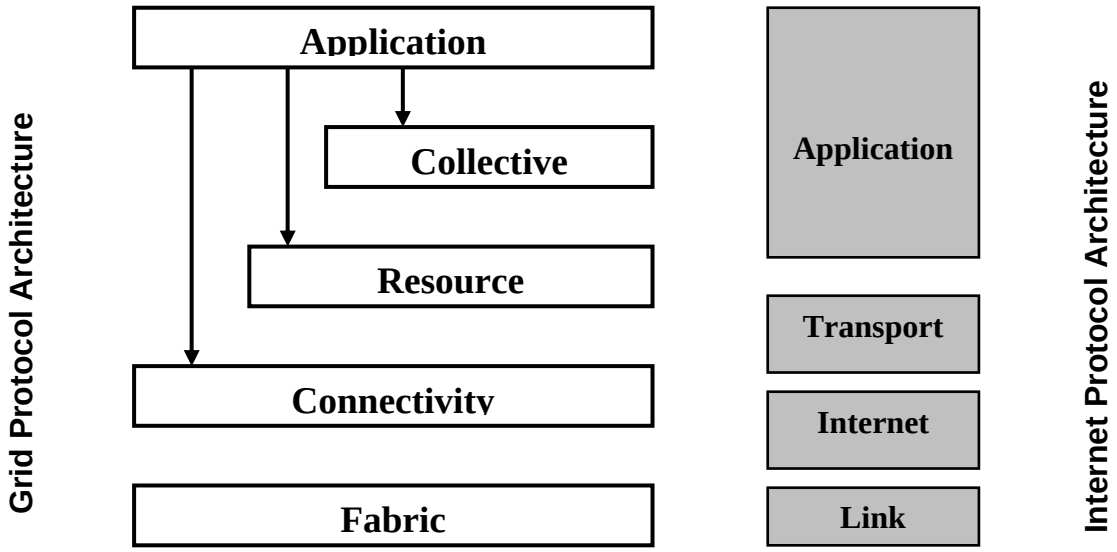


Figure-4.2 The layered Grid architecture and its relationship to the Internet protocol

• ***Computational resources:*** Mechanisms are required for starting programs and for monitoring and controlling the execution of the resulting processes. Management mechanisms that allow control over the resources allocated to processes are useful, as are advance reservation mechanisms. Enquiry functions are needed for determining hardware and software characteristics as well as relevant state information such as current load and queue state in the case of scheduler-managed resources.

• ***Storage resources:*** Mechanisms are required for putting and getting files. Third party and high-performance (e.g., striped) transfers are useful. So are mechanisms for reading and writing subsets of a file and/or executing remote data selection or reduction functions. Management mechanisms that allow control over the resources allocated to data transfers (space, disk bandwidth, network bandwidth, CPU) is useful, as are advance reservation mechanisms. Enquiry functions are needed for determining hardware and software characteristics as well as relevant load information such as available space and bandwidth utilization.

• ***Network resources:*** Management mechanisms that provide control over the resources allocated to network transfers (e.g., prioritization, reservation) can be useful. Enquiry functions should be provided to determine network characteristics and load.

• ***Code repositories:*** This specialized form of storage resource requires mechanisms for managing versioned source and object code: for example, a control system such as CVS.

• ***Catalogs:*** This specialized form of storage resource requires mechanisms for
Implementing catalog query and update operations: for example, a relational database.

Connectivity: Communicating Easily and Securely

The *Connectivity* layer defines core communication and authentication protocols required for Grid-specific network transactions. Communication protocols enable the exchange of data between Fabric layer resources. Authentication protocols build on communication services to provide cryptographically secure mechanisms for verifying the identity of users and resources. Communication requirements include transport, routing, and naming. Authentication solutions for VO environments should have the following characteristics:

• *Single sign on*. Users must be able to "log on" (authenticate) just once and then have access to multiple Grid resources defined in the Fabric layer, without further user intervention.

• *Delegation: A* user must be able to endow a program with the ability to run on that user's behalf, so that the program is able to access the resources on which the user is authorized. The program should (optionally) also be able to conditionally delegate a subset of its rights to another program (sometimes referred to as restricted delegation).

• *Integration with various local security solutions*: Each site or resource provider may employ any of a variety of local security solutions, including Kerberos and Unix security. Grid security solutions must be able to interoperate with these various local solutions. They cannot, realistically, require wholesale replacement of local security solutions but rather must allow mapping into the local environment.

• *User-based trust relationships*: In order for a user to use resources from multiple providers together, the security system must not require each of the resource providers to cooperate or interact with each other in configuring the security environment. For example, if a user has the right to use sites A and B, the user should be able to use sites A and B together without requiring that A's and B's security administrators interact. Grid security solutions should also provide flexible support for communication protection (e.g., control over the degree of protection, independent data unit protection for unreliable protocols, Support for reliable transport protocols other than TCP) and enable stakeholder control over authorization decisions, including the ability to restrict the delegation of rights in various ways.

Resource: Sharing Single Resources

The Resource layer builds on Connectivity layer communication and authentication protocols to define protocols (and APIs and SDKs) for the secure negotiation, initiation, monitoring, control, accounting, and payment of sharing operations on individual resources. Resource layer implementations of these protocols call Fabric layer functions to access and control local resources. Resource layer protocols are concerned entirely with individual resources and hence ignore issues of global state and atomic actions across distributed collections; such issues are the concern of the Collective layer. Two primary classes of Resource layer protocols can be distinguished:

• *Information protocols* are used to obtain information about the structure and state of a resource, for example, its configuration, current load, and usage policy (e.g., cost).

• *Management protocols* are used to negotiate access to a shared resource, specifying, for example, resource requirements (including advanced reservation and quality of service) and the operation(s) to be performed, such as process creation, or data access. Since management protocols are responsible for instantiating sharing relationships, they must serve as a "policy application point," ensuring that the requested protocol operations are consistent with the policy under which the resource is to be shared. Issues that must be considered include accounting and payment. A protocol may also support monitoring the status of an operation and controlling (for example, terminating) the operation. While many such protocols can be imagined, the Resource (and Connectivity) protocols layers

Form the neck of our hourglass model, and as such should be limited to a small and focused set. These protocols must be chosen so as to capture the fundamental mechanisms of sharing across many different resource types (for example, different local resource management systems), while not overly constraining the types or performance of higher-level protocols that may be developed.

Collective: Coordinating Multiple Resources

While the Resource layer is focused on interactions with a single resource, the next layer in the architecture contains protocols and services (and APIs and SDKs) that are not associated with any one specific resource but rather are global in nature and capture interactions across collections of resources. For this reason, we refer to the next layer of the architecture as the *Collective* layer. Because Collective components build on the narrow Resource and Connectivity layer, they can implement a wide variety of sharing behaviors without placing new requirements on the resources being shared

4.5.2 Application Areas of Grid Computing.

These applications fall into the following categories.

Just in Time, or On-demand computing. On-demand computing provides access to specialized resources that are only needed on a short-term or infrequent basis. When such a resource is requested, it is usually needed immediately. However, because access needs are sparse, there might not be sufficient justification to own the resources or even co-locate users with the resources. In a grid environment, many users might share such resources cost-effectively. On-demand computing must dynamically support a potentially large user population and a number of diverse resources. Technology issues with on-demand computing include resource location, scheduling and co scheduling with local resources, code configuration management, security, and payment mechanisms in an open environment.

- *Data-intensive computing*. With the current explosion in information, data-intensive computing becomes an important application area that must be supported by grids,

which involves the synthesis of new information from geographically distributed data sources. A key issue that the grid must solve is scheduling and configuration of high-volume data flows through the multiple levels of network hierarchy.

- *Collaborative computing*. Perhaps the newest application enabled by computational grids is collaborative computing, or the creation of laboratories without walls. The basic goal is to enhance the interactions between humans and computing resources that might be geographically distributed. There are many issues involved in dealing with the real-time requirements human perceptual capabilities and the rich variety of interactions that might take place with the computational resources.
- *High-throughput computing*. High-throughput computing involves executing as many tasks as possible within a given time frame. Often, the tasks are loosely coupled or totally independent. The large number of resources available in a grid provides many spare cycles for throughput computing. Key to the success of high-throughput computing is accurate resource load information and efficient scheduling mechanisms.

Distributed supercomputing. A driving goal of computational grids is to solve problems that cannot currently be solved today on a single system by aggregating computational resources from many sites together. Fundamental issues include parallel algorithm scalability and tight co scheduling of resources from multiple computing sites. Implicit in distributed supercomputing is that the resources are typically very expensive and therefore very limited. Distributed supercomputing might be a component of other grid applications such as on-demand computing or collaborative computing.

CHAPTER –5

REAL TIME TRANSACTION PROCESSING

5.1 what is transaction?

Transaction is an agreement, communication or movement carried out between separate entities or objects often involving the exchange of items of values, such as information goods, services and money.

Transaction is a set of related tasks that either succeed or fail as a unit. Transaction either commits or aborts. These are of various types.

- Database transaction
- Atomic transaction
- Financial transaction

Basically in this paper I am concentrating on distributed transaction and nested transaction.

5.2 Real-time transaction processing.

In a real-time transaction, the deadline refers to the Time by which the transaction must finish or Else undesirable results may occur. Based on the strictness of deadlines, real-time Grid transactions can be classified into three kinds.

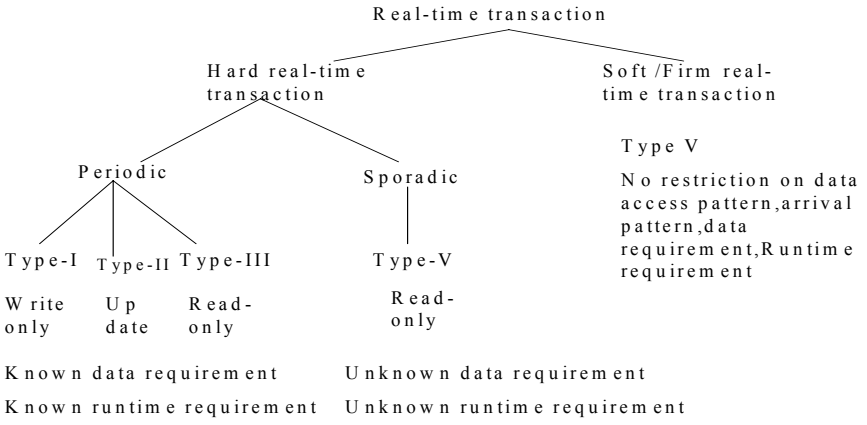


Figure-4.3 classification of real-time transaction

Hard real-time transaction**:** If these transactions miss their deadlines, catastrophic consequences may result.

Firm real-time transaction: It is of no value to complete a firm real-time transaction after its deadline has passed**.** But catastrophic results do not occur after firm real-time transaction passes its deadline.

Soft real-time transaction: Satisfaction of deadline is also the primary performance goal. Unlike a firm real-time transaction, however, there may still be some benefit for completing a soft real-time transaction after its deadline.

ARTTS incorporates fault tolerance into autonomic grid technologies by automatically recovering Systems from various failures. The ARTTS is able to detect and prevent system wide failures for Autonomic grid systems and handle the entire process of a real-time transaction on behalf of Users.

Optimal Grid is an autonomic Grid infrastructure developed in IBM. Using Optimal Grid; the problem owner has no need to concern the partition and deployment of the problem and to know the enlisting of computing nodes. The delivery of the code for various parts of the distributed computing, the run time management of the overall problem and dynamic rebalancing are done automatically. Transaction processing is the effective approach to recovering systems from potential failures. The autonomic Grid must be able to handle exceptions and failures in execution of reliable applications. Benefiting from above efforts, the ARTTS provides the self-protection ability through automating real-time Grid transaction processing to prevent the system from system-wide failures and maintain system consistency and real-time property without intervention of users.

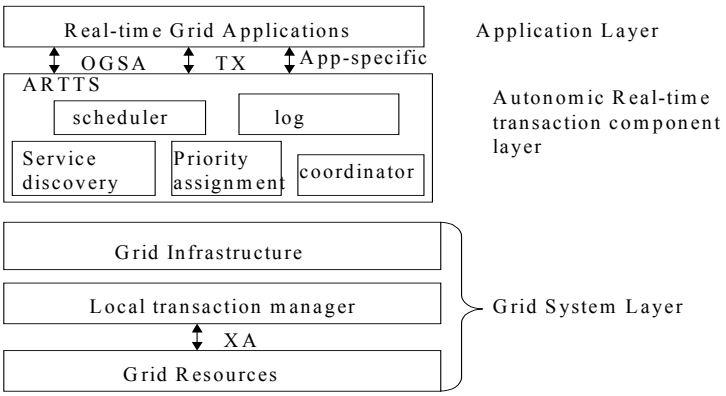
5.2.1 Real-time Grid transaction

Tang et al. (2003a) have discussed coordination of different activities in Grid computing and presented corresponding coordination algorithms for two types of transactions, atomic transaction (AT) and cohesion transaction (CT). The AT, served as coordinating the short-lived transaction, consists of a set of atomic sub transactions that have to commit synchronously. The CT, consisting of atomic sub-transactions or

cohesion sub-transactions, allows some sub-transactions to commit while others fail in order to coordinate the long-lived transaction. The real-time Grid transaction is an extension of the above work in the autonomic Grid environment.

5.2.2 Autonomic real time grid transaction

The ARTTS acts as the real-time Transaction building blocks for making autonomic Grid Systems.



The architecture of the real-time transaction

Figure-5.1

It consists of

- Service Discovery, which discovers appropriate Grid services, i.e., participants, according to application requirements.
- Priority Assignment, which calculates deadlines of (sub) transactions and then assigns priorities to them.
- Coordinator or Participant, which is dynamically created and lives until the end of a global real-time transaction. If the ARTTS receives a request to initiate a real-time transaction, its Scheduler creates a Coordinator to control the global transaction. In case that the ARTTS is required to complete a sub-task, its Scheduler generates a Participant to manage the sub transaction.

- Log, which records the coordination operations and the state information for possible recovery.
- Scheduler, which is responsible for scheduling above modules.
- Interfaces, including (a) the OGSA interfaces for service management such as creating a service instance, (b) the TX interface for transaction management such as starting a transaction, and (c) the application-specific interfaces for specific application purposes.

CHAPTER-6

REAL TIME TRANSACTION PROCESSING FOR AUTONOMIC GRID APPLICATIONS

6.1 Introduction

The future Grid will be an autonomic environment that can not only assist users to share large-scale resources and accomplish collaborative tasks but also self-manage to reduce the users' interventions as much as possible. In such an autonomic Grid environment, the real-time transaction processing is a key and challenging technology to protect systems from various failures. This survey paper presents an autonomic real-time transaction service (ARTTS) that can (1) dynamically discover Grid services as participants to execute specified sub-transactions, (2) coordinate these participants to achieve the real-time and transactional requirements, and (3) assign priorities to schedule concurrent transactions. By handling the potential failures and exceptions autonomically, the ARTTS can facilitate the implementation of real-time Grid transactions and simplify the system management work, which frees users from the complex interference in the autonomic Grid environment.

6.2 Related work

6.2.1 Service discovery

The first step of handing a Grid transaction is to dynamically discover services to execute sub-transactions. The Universal Description, Discovery, and integration (UDDI) define how to publish and discover Web services. Providers of Web services directly publish their services in the UDDI server. Service discovery is an important work in grid transaction, which helps in executing, sub transaction. Service here is of two types

- Transient
- Persistent

The former refers to the services whose instances are created and/or destroyed in runtime and live only for a specified period. Therefore, it is impractical for the UDDI server to manage both creation and registration of millions of remote transient Grid services. This

paper employs two-level registry mechanism to adapt to those transient services. Service descriptions are registered in the Undeserved while its local registry performs creation of a transient service instance.

6.2.2 Transaction processing

Transaction processing has three kinds of roles. Application program, Transaction manager, Resource manager. And two interface XA and TX. Real time scheduler schedule real time transaction using priority assignment policy and resolve data conflict by lock mechanism. Main issue is how to propagate deadlines from global transaction to its sub transaction and how to control concurrent execution of transaction.

6.3 Autonomic Grid computing

Optimal Grid is an autonomic Grid infrastructure developed in IBM. Using Optimal Grid; the problem owner has no need to concern the partition and deployment of the problem and to know the enlisting of computing nodes. The delivery of the code for various parts of the distributed computing, the run time management of the overall problem and dynamic rebalancing are done automatically. Transaction processing is the effective approach to recovering systems from potential failures. The autonomic Grid must be able to handle exceptions and failures in execution of reliable applications. Benefiting from above efforts, the ARTTS provides the self-protection ability through automating real-time Grid transaction processing to prevent the system from system-wide failures and maintain system consistency and real-time property without intervention of users.

6.3.1 Real-time Grid transaction

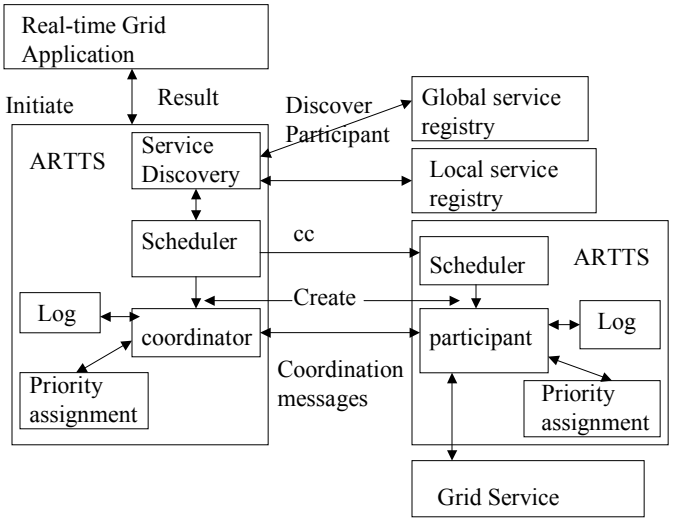
Tang et al. (2003a) have discussed coordination of different activities in Grid computing and presented corresponding coordination algorithms for two types of transactions, atomic transaction (AT) and cohesion transaction (CT). The AT, served as coordinating the short-lived transaction, consists of a set of atomic sub transactions that have to commit synchronously. The CT, consisting of atomic sub-transactions or cohesion sub-

transactions, allows some sub-transactions to commit while others fail in order to coordinate the long-lived transaction. The real-time Grid transaction is an extension of the above work in the autonomic Grid environment.

Transaction processing for the autonomic Grid environment focuses on how to coordinate sub-transactions rather than processing of each individual sub-transaction. The real-time Grid transaction mainly concerns with participant discovery, coordination algorithms, and policies of deadline and priority assignment.

6.3.2 Flow of the real-time Grid transaction processing

In the autonomic Grid environment, a typical real-time transaction processing includes following steps, as shown



The flow the real time Grid transaction processing

Figure-6.1

1. The initial ARTTS initiates a global transaction for a Grid application, discovers and selects satisfactory Grid services to serve as participants, using the Service Discovery module

2. Its Scheduler creates a Coordinator and broadcasts the Coordination Context (CC) messages to all selected remote participants, which create Participant locally and return Response messages to the Coordinator.

3. The created Coordinator and Participants interact to control the transaction execution, including correct completion and failure recovery.

6.3.3 Discovery of participants

In the Grid service environment, any network entity is encapsulated into a service, which is identified by the Grid service handle(s) and reference(s). The goal of discovering participants is to dynamically find the references of service instances. A service discovery model with a two-level registry has been proposed in Tang et al. (2003b). Handles and references of persistent Grid service instances (or handles of factory for transient Grid services) are registered with the Local Service Registry, which publishes service descriptions in the Global Service Registry. The basic steps of service discovery can be described as follows.

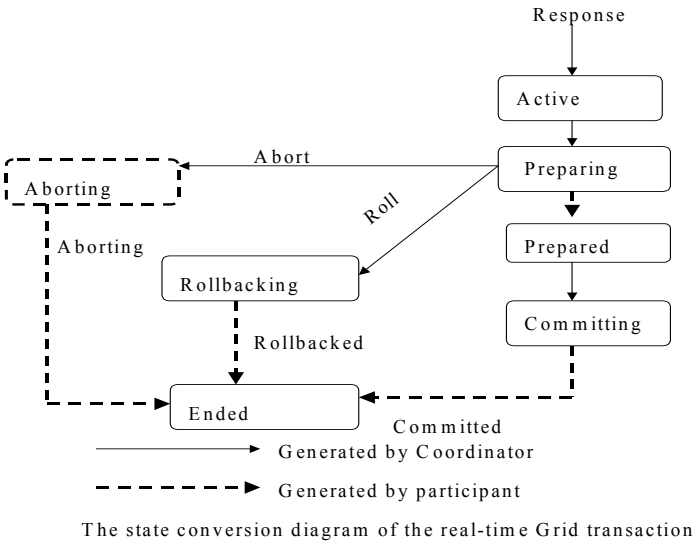
- Query the Global Service Registry to obtain the service description and the handle of their home Local Service Registry.
- Select desirable services based on some policies such as the lowest cost.
- Discover the references of the selected services in the Local Service Registry. For a transient Grid service, its factory service creates the service instances and returns the initial service references.

6.3.4 Coordination of participants

The real-time Grid transaction aims at time-critical Situation, where the most important goal is to maximize the number of transactions that can finish before their deadlines rather than the system throughput. To improve the successful rate, a few functional

alternative services are organized into a functional alternative service group (FASG) to execute the same sub-transaction in parallel. If one member of a FASG can successfully complete before its deadline and reports a Committable message, the FASG is considered committable and other members are aborted. In the preparation phase, each functional alternative service executes a specified sub-transaction in its private work area (PWA). When a service completes the sub transaction successfully before its deadline, it returns a Committable message. Any actual commit occurs only when the global transaction commits. On receipt of an Abort message, the service rollbacks operations taken previously by releasing the PWA. In the commit phase, the Commit message enables the committable participants, which have reported the Committable messages, to commit sub-transactions.

The bellow figure illustrates the state conversion diagram of the Real-time Grid transaction. Solid rectangles indicate the states of both Coordinator and Participants while the dashed rectangle only denotes the state of Participants. Note that the transaction enters the prepared state only after the Coordinator receives a Committable message from each FASG before deadline d (T). Otherwise, the Coordinator sends Rollback messages to all participants.



The state conversion diagram of the real-time Grid transaction

Figure-6.2

6.4. Deadline calculation

Deadline refers to the time by which the transaction must finish or else undesirable results may occur.

Tasks are of two types:

Local task- the task that are executed only at the originating node.

Global task- it consist of series of sub transaction.

The objective is to determine the priorities of the sub transaction so that the percentage of missed deadline is kept as low as possible.

Dead line of local transaction:

$$d(T) = ri + sti + wei$$

d (T)=deadline

r i= arrival time

Sti = slack

Wei = Execution time

Deadline of sub transaction:

A global task is in the form of T=[T1, T2…Tm]

There are basically 4 methods

1. UD-Ultimate deadline

$$dl(Ti) = dl(T)$$

2 ED-Effective Deadline

$$dl(Ti) = dl(T) - \sum_{j=j+1}^{m} Pex(Tj)$$

3. EQS-Equal slack

$$dl(Ti) = ar(Ti) + [dl(T) - ar(Ti) - \sum_{j=i}^{m} pex(Tj)]/(m - i + 1)$$

4. EQF-Equal Flexibility first

$$dl(Ti) = ar(Ti) + pex(Ti) + [(dl(T) - ar(Ti) - \sum_{j=i}^{m} pex(Tj))] * [pex(Ti) / \sum_{j=i}^{m} pex(Tj)]$$

Let X is a transaction

ar(X)—arrival time

dl(X)---deadline

sl(X)—slack

ex(X)—real execution time

pex(X)—Predicted execution time

dl(Ti)—deadline of sub transaction

Earliest deadline first

Highest priority is given to the transaction with closest deadline

6.5 Simulation

Let a model contain k nodes, each node services their tasks according to some real-time scheduling algorithms. Example-EDF.

The transaction manager generates local and global transactions with an independent Poisson stream with an arrival rate varying from 1 to 80 transactions per second.

Evaluation time, deadline of transaction follows a uniform distribution.

No of transaction resources access is atleast one main memory database is taken for simplicity.

First step of simulation consisted in measuring the global miss ratio for different scheduling algorithm, let EDF and FCFS.GMR is defined as The total number of transaction that miss their deadlines compared to the total number of transactions accepted by the algorithm.

Table 1

System Parameter

Parameter	Description	Value
We_i	Execution time	30 to 300 ms
sf_i	Slack factor: $st_i = sf_i * We_i$ $d_i = r_i + We_i (1 + sf_i)$	3 to 5
ג	Transaction arrival rate per second	1 to 80 transaction per second
NR	Number of resources	20

6.6 Results

We found that the EDF algorithm have less global miss ratio in compare to FCFS.both the algorithm performs almost identically until number of transaction per second grows to

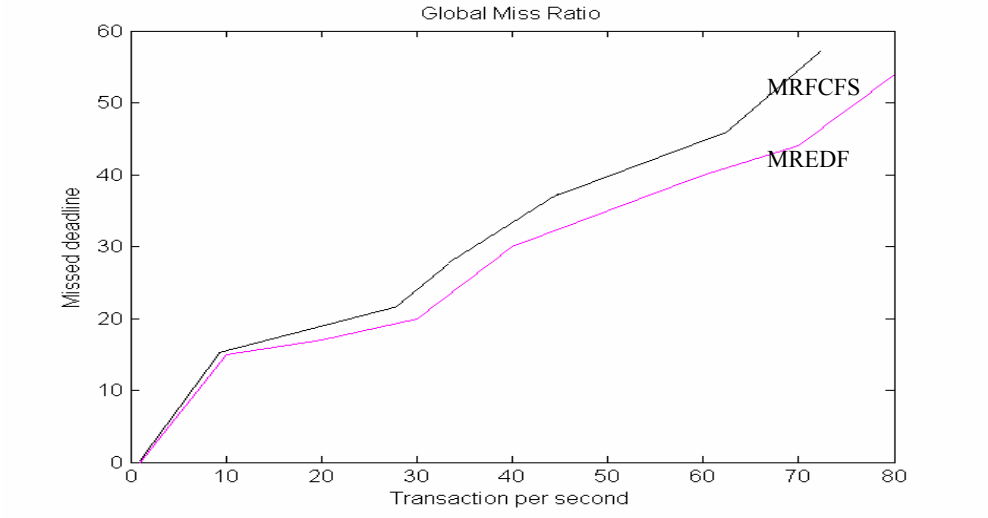


Figure 6.3

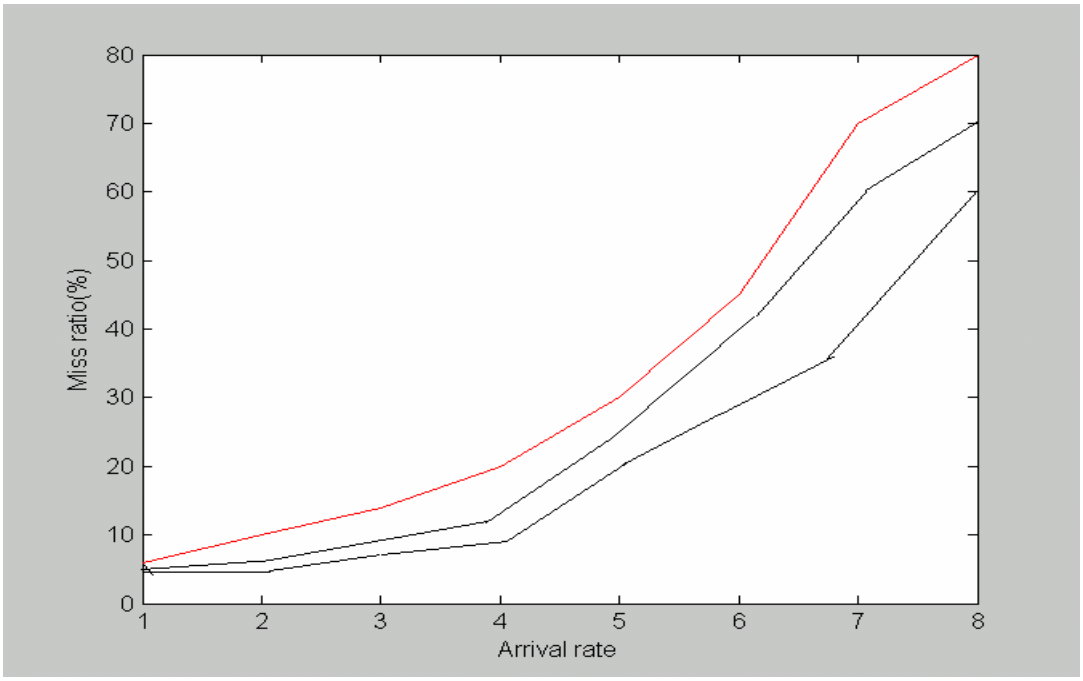


Figure 6.4 Miss ratio in different workload

In the above figure value of ג is 1,4,7 from bottom to top. With increase in workload miss ratio first decreased and then rose. The reason was each sub transaction acted as a unit to compete for resources so that more workload the more system resource they consumed. So more transaction missed their deadlines, as they could not get enough resource in time.

6.7 Conclusion and future works

In this paper we basically concentrate on the autonomic systems. We discuss about autonomic real time transaction service. How this service dynamically discover grid services to execute specified sub-transactions. Dynamically assign priorities for scheduling concurrent transaction. As a result, by handling the entire transaction process on behalf of users automatically it facilitate the implementation of real-time and transactional grid application to provide self-protection function and simplify the management work.

The future work will be to make the autonomic application more practical and satisfy all the four properties for solving various issues in the autonomic grid environment. The future work will focus on combining security measure with our work's the overall grid environment will get authentication, authorization and communication protection.

Bibliography

[1] Feilong Tang, Minglu Li, and Joshua Zhexue Huang. "Real-time transaction Processing for autonomic Grid applications,"Engineering Application of Artificial Intelligence 17(2004), pp.799-807, China, 2004.

[2] Xiaolong Jin, and Jiming Liu, "Characterizing autonomic task distribution and Handling in grids," Engineering Application of Artificial Intelligence 17(2004), Pp.809-823, Hong Kong, 2004.

[3] Rainer Unland, and Huaglory Tianfield, " Towards Autonomic computing Systems," Engineering Application of Artificial Intelligence 17(2004), pp.689-699, Germany, 2004.

[4] Eser Kandogan, John Bailey Rob Barrett, and Paul p. Maglio, "Usable autonomic computing systems: the system administrators' perspective," In Advance Engineering Informatics *19*, pp.213-221, Nov 2005.

[5] Roy Sterritt, and Dave Bustard, "Towards an Autonomic computing Environment," Proceeding of IEEE 14th international workshop on Database and Expert systems applications (DEXA '03), 2003.

[6] Ricardo M.Bastos, Flavio M.de Oliveira, and Jase Palazzo M.de Oliveira, "Autonomic computing approach for resource allocation,"Expert systems with applications 28(2005), pp.9-19, Brazil, 2005.

[7] Andrea Baldini, Alfredo Benso, and Paolo Prinetto. "A dependable autonomic computing environment for self-testing of complex heterogeneous systems,"Electronics notes in Theoretical Computer science 116(2005), pp.45-57, Italy, 2005.

[8] Jeffrey O.Kephart, and David M.chess, "The vision of Autonomic Computing,"IEEE computer society, Jan 2003, pp.41-49.

[9] Paul Lin, Alexander MacArthur, and John Leaney, "Defining Autonomic Computing: A software engineering prospective,"Proceedings of the 2005 IEEE Australian Software engineering conference (ASWEC '05), 2005.

[10] Steve R.white, James E.Hanson, Ian Whally, David M.chess, and Jeffrey O.Kephart, "An Architectural Approach to Autonomic Computing," Proceedings of IEEE International conference on Autonomic Computing (ICAC'04), 2004.

[11] E.Grishikashvili Pereira, R.Pereira, and A.Taleb-Bendiav, "Performance evaluation for self healing Distributed services and fault detection mechanisms," Journals of Computer and System Science, 2006.

[12] Yuan-Shan Dai, "Autonomic Computing and Reliability Improvement," Proceedings of the 8th IEEE International Symposium on Object-Oriented Real-Time distributed computing, 2005.

[13] Shimon Whiteson, and Peterstone, "Towards Autonomic Computing: Adaptive network routing and scheduling," Proceeding of IEEE International Conference Autonomic Computing, 2004.

[14] L.Baccouche. , "Scheduling Muli-class real-Time Transactions: A performance Evaluation,"Enformatika Society, vol 6,pp.249-252, 2005.

[15] Ben kao and hector Garcia-Molina, "deadline assignment in a Distributed soft real-Time System,"IEEE Transaction on parallel and distributed systems. Vol 8,Dec 1997.