An Acquaintance of Checkpointing Provision in Mobile Computing

D.R.S.SWETHA
LECTURER IN COMPUTER SCIENCE,
COMPUTER SCIENCE,
Dr. LANKAPALLI BULLAYA COLLEGE, VISAKHAPATNAM, INDIA

Abstract: This Checkpointing is more decisive in Mobile Grid MoG computing systems and modules than in their conventional and bidirectional transmission wired counterparts due to host mobility, uncountable, less reliable wireless networks, frequent disconnections and variations in mobile computing systems. This paper resolute the global optimal checkpoint provisioning to be NP-complete and so consider Reliability Driven middleware, employing suburbanized Quality of Service (QoS) -aware heuristics, to construct superior checkpointing arrangements efficiently. With Reliability Driven (ReD), an MH (mobile host) simply sends its checkpointed data to one selected neighboring MH, and also serves as a stable point of storage for checkpointed data received from a single approved neighboring MH. ReD works to maximize the probability of checkpointed data recovery during job execution, increasing the likelihood that a distributed application, executed on the MoG, completes without sustaining an unrecoverable failure. In this project the efficiency of the system, Packet transmission time and Receiving time is also calculated. An approach to implement a Checkpointing arrangement (RCA) middleware, a QoS-blind comparison protocol is used in this project.

Index Terms **- Checkpointing, Mobile Grid Systems (MoGS), Quality of Service (QoS), Reliability Driven, Implementation & Testing**

I. INTRODUCTION

GRID computing systems have seen their widespread adoption lately in not only academia but also in industry, as evidenced by commercial offerings from Sun [1], HP [2], and IBM [3], among others. While most existing Grids refer to clusters of computing and storage resources which are wire-interconnected for offering utility services collaboratively, Mobile Grids (MoGs) are receiving growing attention and expected to become a critical part of a future computational Grid involving mobile hosts to facilitate user access to the Grid and to also offer computing resources [4]. A MoG can involve a number of mobile hosts (MHs), i.e., laptop computers, cell phones, PDAs, or wearable computing gear, having wireless interconnections among one another, or to access points. Indeed, a recent push by HP to equip business notebooks with integrated global broadband wireless connectivity [5], [52], has made it possible to form a truly mobile Grid (MoG) that consist of MHs providing computing utility services collaboratively, with or without connections to a wired Grid.

Current trends toward powerful multicore processors, efficient, small flash memory devices, and wireless technologies, such as IEEE 802.16 WiMAX (which are capable of delivering 10 Mbps or more over distances of miles), are seen as technological enablers of the practical MoG, while models for compensation, accounting, and regulation of these systems are being developed. Other researchers proposed a middleware using mobile agents for secure MoG services, addressed the heterogeneity concerns of this technology, and provided compatible interfaces to the Globus Toolkit [3].

1.1 Overview:

Checkpointing saves intermediate data and machine states periodically to reliable storage during the course of job execution. Various checkpointing mechanisms have been pursued for distributed systems (whose computing hosts are wire-connected) [6], [7], [8], [9], [10], [11], [12], [13]. However, they are not suitable for the MoG because their checkpointing arrangements 1) are relatively immaterial as checkpointed data from hosts can be stored at a designated server or servers, since connections to a server are deemed reliable, of high bandwidth, and of low latency, and 2) fail to deal with link disconnections and degrees of system topological dynamicity. In contrast, a MoG highly desires its checkpointed data to be kept all at neighboring MHs rather than remote ones who require multiple, relatively unreliable, hops to transmit checkpoints and to reach checkpointed data when it is needed. Earlier wireless checkpointing methods stored checkpointed information at fixed, stationary hosts, on the wired Grid (i.e., base stations, (BSs)) via access points [4], [5], [9], [2]. This is suitable only for systems where every MH can reach a BS in one hop, an unlikely scenario if the MoG is to be realized practically.

1.2 Checkpointing:

Checkpointing is a technique for inserting fault tolerance into computing systems. It basically consists of storing a snapshot of the current application state, and later on, use it for restarting the execution in case of failure.Itsaves intermediate data and machine states periodically to reliable storage during the course of job execution.

1.3 Problem Definition

The choice of checkpointing arrangement has significant bearing upon MoG system reliability and thus QoS, Quality of Service (namely, the probability that an application will complete feasibly within the bounds of the client's a priori specified limit in terms of time frame, reliability, etc.) expected by a user's application. Different arrangements yield differing

probabilities that the checkpointed data will survive and be recoverable in the presence of host failure, or more likely and more frequently, link failure or inadvertent and intermittent disconnection of a host or hosts from the MoG. An efficient arrangement thus aims to identify superior assignments of providers and consumers of checkpointing services, among all MHs within the MoG in order to ensure collaborative job execution can complete reliably with smaller probability of experiencing unrecoverable failures. In order to avoid such failures it is shown that the globally optimal checkpointing arrangement is an NP complete problem [3]. ReD is thus based upon a heuristic formulation that attains arrangements of superior reliability. Its efficient convergence is promoted by a derived and supportive, simple clustering algorithm, allowing concurrent operation on small clusters of hosts individually, rather than on a single large global system.

II. LITERATURE SURVEY

From Cluster Grids to- ward Mobile Collaborative Business Grids Based on the definition of Grid Types in the term Grid can be seen as a summary term for at least the following different flavours of "Grids".

2.1 Distributed Enterprise Grids:

As current security models available in commercially supported Grid toolkits or even recent developments in research do not meet the expectations of commercial environ- ments for resource sharing across organisational boundaries many deploy- ments of Grids as of today are operating behind companies firewalls. These Intra Grids may however be already geographically dispersed and are typically connected with standard network equipment.

2.2 Utility Grid Services:

This type of Grids can be seen as the natural next step in the evolution and is similar to Intra Grids but is additionally able to be extended on demand e.g. with computational resources offered by a third party as needed. This definition assumes that resource providers sell their resources e.g. for computation as a utility.

2.3 Managed Hosted Grids:

In extension to the rather limited view in [7] defining this Grid Type as an Intra Grid operating only within a single company we think this type of third party Managed Grids will emerge as well for the case where a resource owner wants to offer its resources as an utility or even as participant in collaborative business Grids.

2.4 An Adaptive Checkpointing Protocol

Numerous mathematical approaches have been proposed to determine the optimal checkpoint interval for minimizing total execution time of an application in the presence of failures. These solutions are often not applicable due to the lack of accurate data on the probability distribution of failures. Most current checkpoint libraries require application users to define a fixed time interval for checkpointing. The checkpoint interval usually implies the approximate maximum recovery time for single process applications. However, actual recovery time can be much smaller when message logging is used. Due to this faster recovery, checkpointing may be more frequent than needed and thus unnecessary execution overhead is introduced. In this paper, an adaptive checkpointing protocol is developed to accurately enforce the user-defined recovery time and to reduce excessive checkpoints. An adaptive protocol has been implemented and evaluated using a receiver-based message logging algorithm on wired and wireless mobile networks. The results show that the protocol precisely maintains the user-defined maximum recovery times for several traces with varying message exchange rates. The mechanism incurs lour overhead, avoids unnecessary checkpointing, and reduces failure free execution time.

2.5 Recoverable Mobile Environment

The mobile wireless environment poses challenging problems in designing fault-tolerant systems because of the dynamics of mobility, and limited bandwidth available on wireless links. Traditional fault-tolerance schemes, therefore, cannot be directly applied to these systems. Mobile systems are often subject to environmental conditions which can cause loss of communications or data. Because of the consumer orientation of most mobilesystems, run-time faults must be corrected with minimal (if any) intervention from the user. The fault-tolerance capability must, therefore, be transparent to the user.It portrays the limitations of the mobile wireless environment, and their impact on recovery protocols.

2.6 Related Work

The model of a service-oriented,market-based Grid is presented. In this environment,the most challenging economic-related problems is pointed, which seem to prevent users from fully exploiting the potential value of the Grid. At any time during job execution, a host or link failure may lead to severe performance degradation or even total job abortion, unless execution checkpointing is incorporated. Checkpointing forces hosts involved in job execution to periodically save intermediate states, registers, process control blocks, messages, logs,etc, to stable storage.

III. SYSTM ANALYSIS

3.2 Functional Requirements

3.2.1 Introduction

It provides an overview of the system and some additional information to place the system in context. A software requirements specification (SRS) is a comprehensive description of the intended purpose and environment for software under development. The SRS fully describes what the software will do and how it will be expected to perform.

An SRS minimizes the time and effort required by developers to achieve desired goals and also minimizes the development cost. A good SRS defines how an application will interact with system hardware, other programs and human users in a wide variety of real-world situations. Parameters such as operating speed, response time, availability, portability, maintainability, footprint, security and speed of recovery from adverse events are evaluated.

3.2.2 Purpose

It provides an overall description of the checkpointing arrangement, its purpose. Reference the system name and identifying information about the system to be implemented.

3.2.3 Scope

It discusses about the Reliability Driven(ReD) and the checkoint arrangement to be NP- Complete and also about the packet transmission.

3.3 EXISTING SYSTEM:

Check pointing is a technique for inserting fault tolerance into computing systems. It basically consists of storing a snapshot of the current application state, and later on, uses it for restarting the execution in case of failure. Check pointing is more crucial in MoG systems than in their conventional wired counterparts due to host mobility, dynamicity, less reliable wireless links, frequent disconnections and variations in mobile systems.Grid computing is a term referring to the combination of computer resources from multiple administrative domains to reach common goal.

3.3.1 Drawbacks of the Existing System:

Existing work has considered various forms of coordinated or uncoordinated checkpointing on large systems but failed to address checkpointing arrangement and how to store checkpoint information and who is given priority to store it. In existing system the losses and performance are unpredictable.

3.4 Proposed System:

An automatic decentralized, QoS-aware middleware for checkpointing arrangement in Mobile Grid (MoG) computing systems is proposed. It determines the globally optimal checkpoint arrangement to be NP-complete and so consider Reliability Driven (ReD) middleware, employing decentralized QoS-aware heuristics, to construct superior checkpointing arrangements efficiently.With ReD, an MH (mobile host) simply sends its checkpointed data to one selected neighboring MH, and also serves as a stable point of storage for checkpointed data received from a single approved neighboring MH.ReD works to maximize the probability of checkpointed data recovery during job execution.The data losses and time consumption were reduced by using probability distribution as well as increases the performance evaluation.

3.5 Requirement Analysis:

Requirement analysis is a software engineering task that bridges the gap between system level requirements engineering and software design. Requirements engineering activities results in the specification of software's operational characteristics (function, data, and behavior), indicate software interface with other system elements, and establish constraints that software must meet. Requirement analysis involves so many activities mainly gathering requirements, define problem statement. In the case of object oriented analysis the process is varies.

Use Case: A use case describes the behaviour of a system. It is used to structure things in a model. It contains multiple scenarios, each of which describes a sequence of actions that is clear enough for outsiders to understand.

Actor: An actor represents a coherent set of roles that users of a system play when interacting with the use cases of the system. An actor participates in use cases to accomplish an overall purpose. An actor can represent the role of a human, a device, or any other systems.

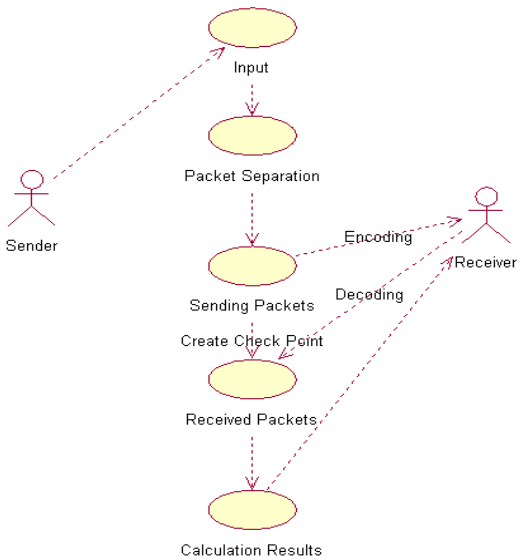

Fig 3.1: Use case view

3.5.1 Sequence Diagram:

Class diagrams and object diagrams represent static information. In a functioning system, however, objects interact with one another, and these interactions occur over time. The UML sequence diagram shows the time-based dynamics of the interaction. Object can be viewed as an entity at a particular point in time with a specific value and as a holder of identity that has different values over time. Associations among objects are not shown. When you place an object tag in the design area, a lifeline is automatically drawn and attached to that object tag.

A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner. For instance, the UML 1.x diagram on the right describes the sequences of messages of a (simple) restaurant system. This diagram represents a Patron ordering food and wine, drinking wine then eating the food, and finally paying for the food. The dotted lines extending downwards indicate the timeline. Time flows from top to bottom. The arrows represent messages (stimuli) from an actor or object to other objects. For example, the Patron sends message 'pay' to the Cashier. Half arrows indicate asynchronous method calls. The UML 2.0 Sequence Diagram supports similar notation to the UML 1.x Sequence Diagram with added support for modeling variations to the standard flow of events.The below figure (3.2) depicts the sequence diagram of the system.

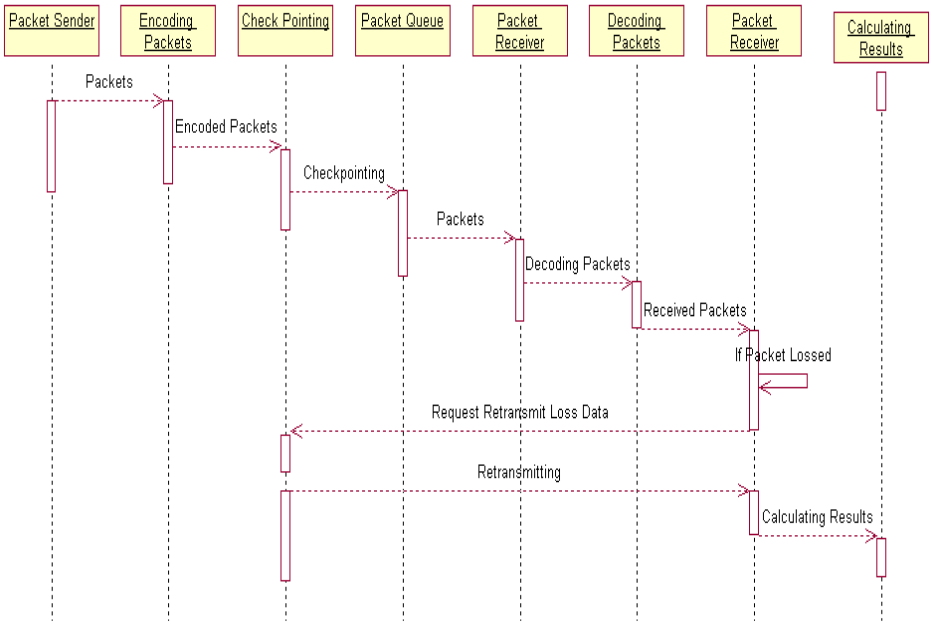

Fig 3.2: Sequence diagram of the system

3.5.2Activity Diagram:

The activities that occur within a use case or within an object's behavior typically occur in a sequence, as in the steps listed in the preceding subsection. Figure 8 shows how the UML activity diagram.

Activity diagram address the dynamic view of a system. They are especially important in modeling the function of system and emphasize the flow of control among objects.

These activity diagrams show how the use-cases interact with the system and interface. The User starts by initially interacting with the system. The main page is then rendered by the system and it is displayed by the interface, which the user can view. From here the user can click on a link, scroll or close the system.

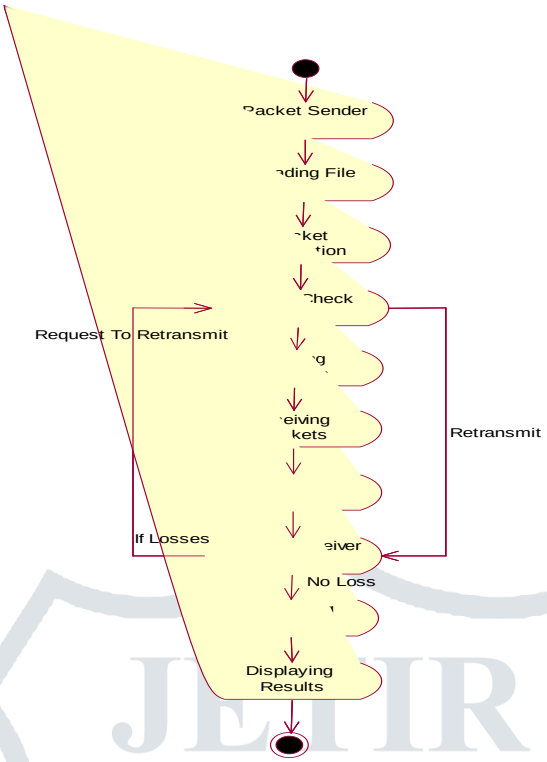

Fig: 3.3: Activity diagram of the system

3.5.3 State Chart Diagram:

State diagrams are used to give an abstract description of the behavior of a system. This behavior is analyzed and represented in series of events, that could occur in one or more possible states. Hereby "each diagram usually represents objects of a single class and tracks the different states of its objects through the system.

State diagrams can be used to graphically represent finite state machines. This was introduced by Taylor Booth in his 1967 book "Sequential Machines and Automata Theory". Another possible representation is the State transition table.The below figure (3.4) depicts the state chart diagram of the system.

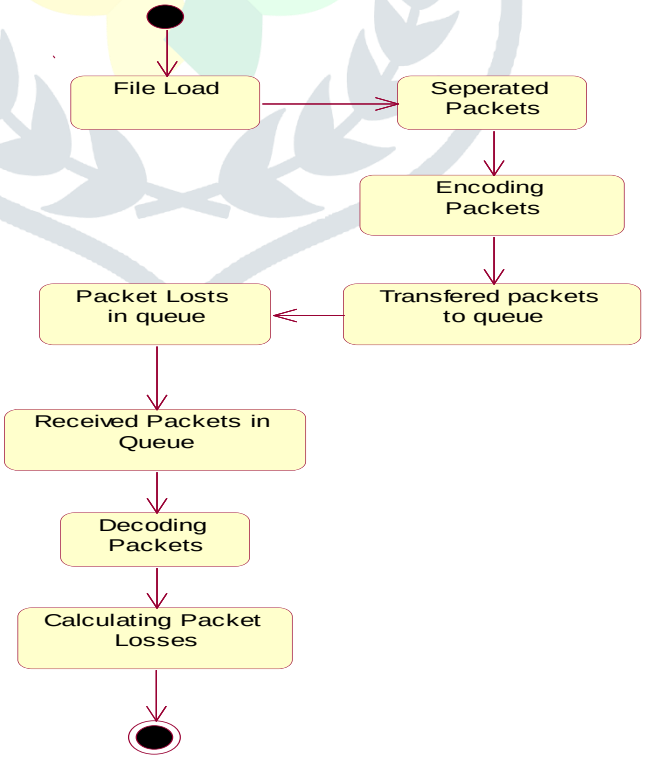

Fig 3.4 State chart diagram of the system

3.5.4 Collaboration Diagrams:

The elements of a system work together to accomplish the system's objectives, and a modeling language must have a way of representing this. Rather than represent time in the vertical dimension, this diagram shows the order of messages by attaching a number to the message label. Both the sequence diagram and the collaboration diagram show interactions among objects. For this reason, the UML refers to them collectively as interaction diagrams.

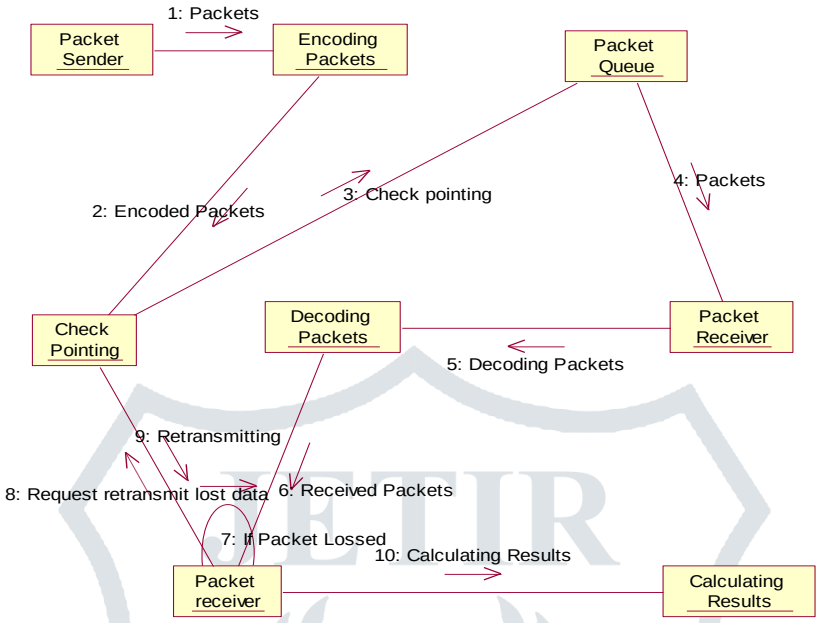

Fig 3.5: Collaboration diagram of the system

3.5.5 Component Diagram:

Components are wired together by using an *assembly connector* to connect the required interface of one component with the provided interface of another component. This illustrates the *service consumer - service provider* relationship between the two components.

An *assembly connector* is a "connector between two components that defines that one component provides the services that another component requires. An assembly connector is a connector that is defined from a required interface or port to a provided interface or port.The below figure (3.6) depicts the component diagram of the system.

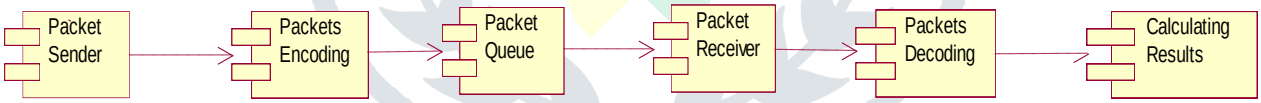

Fig 3.6: Component diagram of the system

IV. SYSTEM DESIGN

4.1 Architectural Diagram:

Architecture is the hierarchical structure of a program components (modules), the manner in which these components interact and the structure of data that are used by that components.

4.1.1 System Architecture:

System Architecture describes "the overall structure of the system and the ways in which that structure provides conceptual integrity".

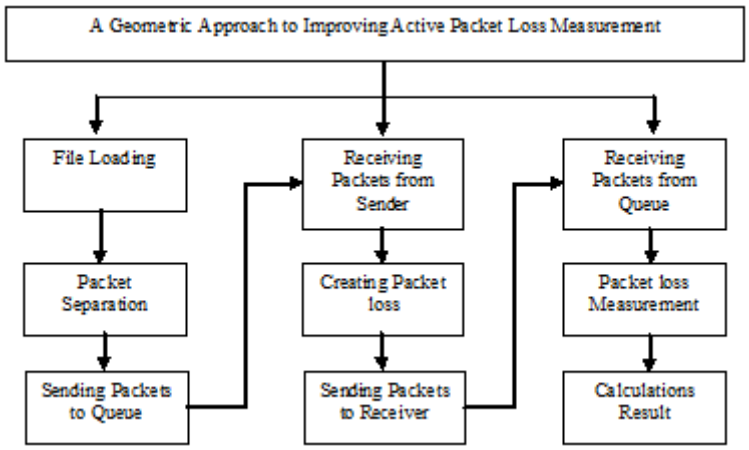

Fig 4.1: System Architecture

The above diagram (fig 4.1) refers to the system architecture that describes the overall stucture of the system. Intially the sender sends an input that is the file is loaded after that the packet separation is done. Once the packets is done, the packets are sent to Queue,the the receiver receives the packets from the sender. After that the packet loss is created and packets are sent to the receiver. It receives packets from the queue and the packet loss measurement is calculated and finally the result is shown.

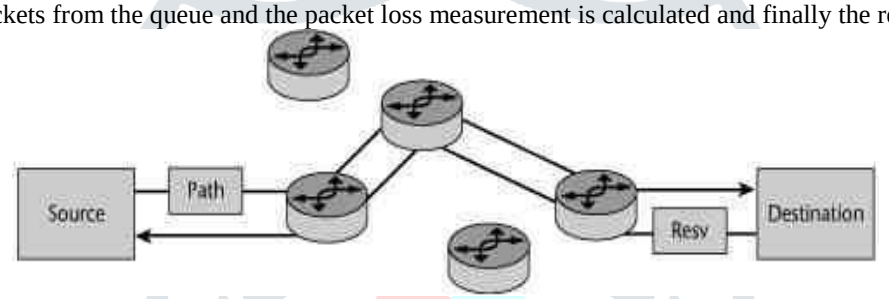

Fig 4.2: Block Diagram

The above figure(4.2) depicts about the block dagram of the system that is it contains both source and destination and the path is specified for the packets to be transferred from source to the destination. The packets are received by the receiver from the queue which is sent by the sender at the source to the destination.

4.2 UML Diagrams:

UML stands for Unified Modeling Language. UML is a modeling language for specifying, visualizing, constructing, and documenting the artifacts of a system intensive process. This object-oriented system of notation has evolved from the work of Grady Booch, James Rumbaugh, Ivar Jacobson.

The primary goals in the design of the UML are:

- Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
- Provide extensibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development processes.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of the OO tools market.
- Support higher-level development concepts such as collaborations, frameworks, patterns and components.

There are three classifications of UML diagrams:

4.2.1 Behavior diagrams*:* A type of diagram that depicts behavioral features of a system or business process. This includes activity, state machine, and use case diagrams as well as the four interaction diagrams.

4.2.2 Interaction diagrams: A subset of behavior diagrams which emphasize object interactions. This includes communication, interaction overview, sequence, and timing diagrams.

4.2.3 Structure diagrams: A type of diagram that depicts the elements of a specification those are irrespective of time. This includes class, composite structure, component, deployment, object, and package diagrams.

A large number of libraries with ready-made objects for UML diagrams and task-oriented templates let you create UML diagrams of any complexity without effort.

- Activity diagram
- Use case diagram
- Sequence diagram
- Collaboration diagram
- Statechart diagram
- Class diagram
- Component diagram

- Deployment diagram

UML (Unified Modeling Language) is a general-purpose modeling language used to represent the structure of complex software in a visual form, and employed in software engineering. UML diagrams are also efficient for documenting complex computer systems and software. Such visual models facilitate communication between the customer, system analysts and programmers, who write the source code. In addition, it's much easier for new programmers to understand the source code when a detailed UML diagram of it is available. Previously, when a programmer resigned a part of his work was gone with him because the code-creating process took place exclusively in his head. Now a newcomer can understand and get familiar with someone else's code without much trouble.

Programming languages may use operations and methods which are basically same, though vary by names and graphical notations. The UML language defines standards not only for operations and methods of programming languages, but also for their terminology. So, using UML diagrams for visual modeling will help you to improve the final software products, simplify the process of its creation and meet the deadlines.

4.3 Class Diagram:

A total of 5 classes isindentified. A Lexer class and a Parser class - which comprise the Analyser package – a ParsedTreeStructure class, a Renderer class and a Frontend class.The Lexer's job is to build a set of tokens from a source file. The Parser uses these tokens built and deciphers their types. It then builds the tokens seen into nodes and parses them to the ParsedTreeStructure class, where a tree structure of nodes is stored. This tree is then used by the Renderer class to form a model of the page, which is in turn, is used by the Frontend in order to display the final rendered page. The below figure (4.3) depicts the class diagram of the system.

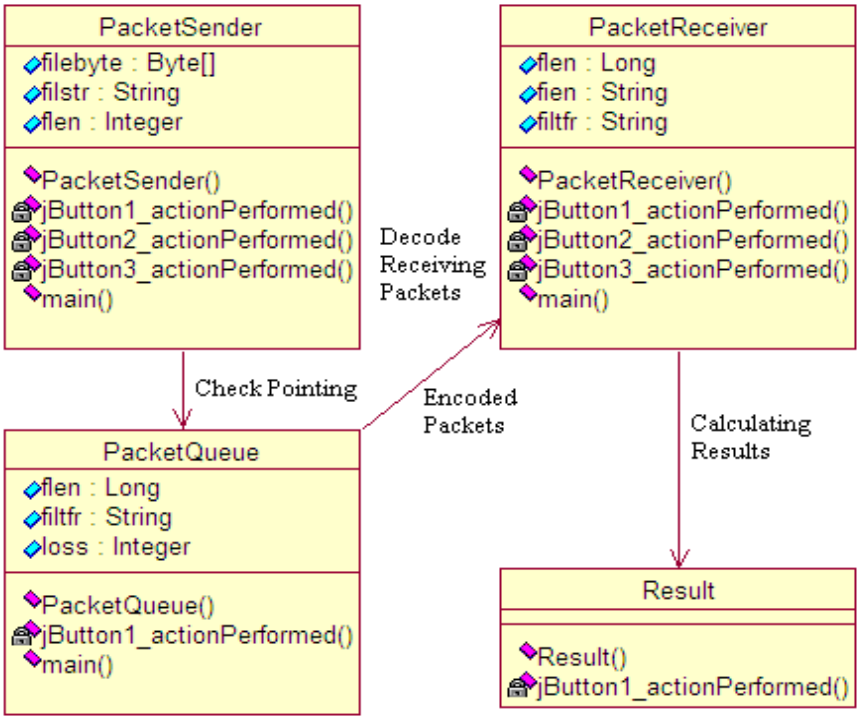

Fig 4.3: Class diagram of the system

V. IMPLEMENTAION

5.1 Introduction

Implementation is the stage where the theoretical design is turned into a working system. The most crucial stage in achieving a new successful system and in giving confidence on the system for the users that will work efficiently and effectively. The system will be implemented only after through testing and if it is found to work according to the specification. It involves careful planning, investigations of the current system.

Overview of Software Used

This application is developed and executed with the jdk1.6.0-07 handling the J2SE java part with User interface Swing component. Java is robust, object oriented, multi-threaded, distributed and secure and platform independent language. It has wide variety of package to implement our requirement and number of classes and methods can be utilized for programming purpose. These features make the programmer's to implement to require concept and algorithm very easier way in Java.

5.2 Modules

1. ReD's Heuristic Basis
   - Packet sender
   - Packet queue
   - Packet receiver
2. Testing
3. Checkpoint

4. Performance Evaluation

5.2.1 Module Description:

5.2.1.1 ReD's Methodology:

An executing host is considered to be in "failure," if wireless connections to all of its neighbors are disrupted temporarily or permanently, resulting in its isolation and inability toachieve timely delivery of intermediate or final application results to other hosts. Executing MHs with poor connectivity, have greater likelihood of experiencing failure than do those with greater connectivity and are thus in greater need of checkpointing to the best, most reliably connected providers We conclude from Fig. 5.1 that no unrecoverable failureoccurs under the following two cases:

Case 1: No failure at participating hosts (i.e., no isolationof any host when its results are required). A givenapplication will run to completion without restarting orrewinding. This is true whether or not checkpointing isdone.

Case 2: Failure of a participating host (isolation). Theapplication will run to completion, with recovery assistance,provided that upon a host failure, its latest savedcheckpoint data exists on the provider host, which is notitself isolated and can then be accessed by the MoG duringthe recovery process. This is possible only if 1) the link forsending checkpoints to the provider host did not fail whenbeing used to checkpoint, and 2) the provider host did notfail (i.e., did not itself become isolated from the rest of theMoG) during the attempt by the MoG to access savedcheckpointed data upon recovery.

Our checkpoint arrangement protocol, ReD, makes use ofthe underpinnings in both cases, resulting in the basis modelFig. 5.1 depicts. As long as all hosts running the distributedapplication have high connectivity (i.e., low likelihood ofseparation from the MoG), robust safe storage or reliabletransmission to safe storage is not really needed. Only poorly connected hosts (e.g., hosts on the fringes of the MoG) need robust safe storage and reliable transmission to that safestorage.

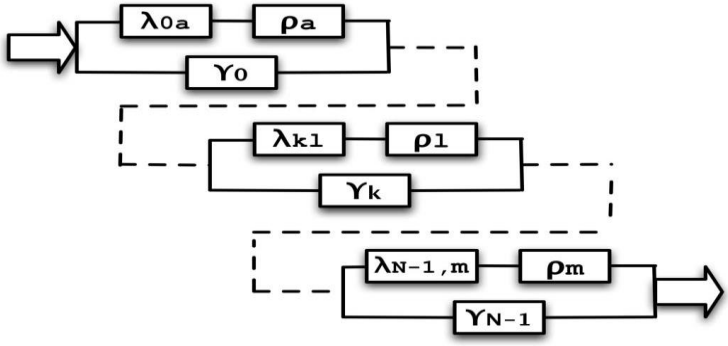

Fig 5.1: ReD reliability model.

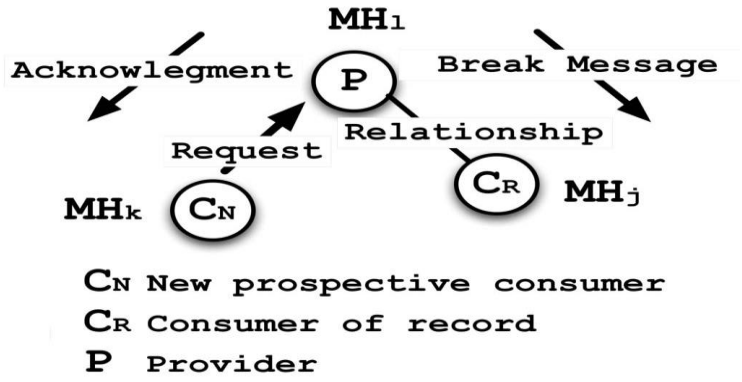

Fig. 5.2: Protocol function.

IV. TESTING DISCUSSION AND RESULTS

6.1 Introduction

Software testing is the process used to assess the quality of computer software. Software testing is an empirical technical investigation conducted to provide stakeholders with information about the quality of the product or service under test, with respect to the context in which it is intended to operate.

Software testing is a process of *verifying* and *validating* that a software application or program. Software testing

1. Meets the business and technical requirements that guided its design and development, and

2. Works as expected.

Software testing also identifies important *defects*, flaws, or errors in the application code that must be fixed. The modifier "important" in the previous sentence is, well, important because defects must be categorized by severity. The quality assurance aspect of software development—documenting the degree to which the developers followed corporate standard processes or best practices—is not addressed in this paper because assuring quality is not a responsibility of the testing team. The testing team cannot improve quality; they can only measure it, although it can be argued that doing things like designing tests before coding begins will improve quality because the coders can then use that information while thinking about their designs and during coding and debugging.

Software testing has three main purposes: verification, validation, and defect finding.

- The *verification* process confirms that the software meets its technical specifications. A "specification" is a description of a function in terms of a measurable output value given a specific input value under specific preconditions. A simple specification may be along the line of "a SQL query retrieving data for a single account against the multi-month account-summary table must return these eight fields <list> ordered by month within 3 seconds of submission."
- The *validation* process confirms that the software meets the business requirements. A simple example of a business requirement is "After choosing a branch office name, information about the branch's customer account managers will appear in a new window. The window will present manager identification and summary information about each manager's customer base: <list of data elements>." Other requirements provide details on how the data will be summarized, formatted and displayed.
- A *defect* is a variance between the expected and actual result. The defect's ultimate source may be traced to a fault introduced in the specification, design, or development (coding) phases.

6.2 Factors Of Testing

Testing can involve some or all of the following factors.

- Business requirements
- Functional design requirements
- Technical design requirements
- Regulatory requirements
- Programmer code
- Systems administration standards and restrictions
- Corporate standards
- Professional or trade association best practices
- Hardware configuration
- Cultural issues and language differences

Software testing is not a one person job. It takes a team, but the team may be larger or smaller depending on the size and complexity of the application being tested.

The release of a new application or an upgrade inherently carries a certain amount of risk that it will fail to do what it's supposed to do. A good test plan goes a long way towards reducing this risk. By identifying areas that are riskier than others we can concentrate our testing efforts there.

6.3 REDUCE RISK WITH A TEST PLAN

Component	Description	Purpose
Responsibilities	Specific people who are and their assignments	Assigns responsibilities and keeps everyone on track and focused
Assumptions	Code and systems status and availability	Avoids misunderstandings about schedules
Test	Testing scope, schedule, duration, and prioritization	Outlines the entire process and maps specific tests
Communication	Communications plan—who, what, when, how	Everyone knows what they need to know when they need to know it
Risk Analysis	Critical items that will be tested	Provides focus by identifying areas that are critical for success
Defect Reporting	How defects will be logged and documented	Tells how to document a defect so that it can be reproduced, fixed, and retested
Environment	The technical environment, data, work area, and interfaces used in testing	Reduces or eliminates misunderstandings and sources of potential delay

Table 6.1: Reducing risk with the test plan

6.4 SYSTEM TESTING

Software testing forms an important activity of the software development. Software testing identifies errors on early stage. A planned testing identifies the difference between the expected results and the observed results. The main objective of the software testing is to find errors. A successful testing is our that uncovers, as many as yet UN discovered errors, which helps to make the software more rugged and reliable. It is applied at the different levels in the SDLC, but testing is done in different nature and has different objectives at each level. Software testing is a critical element or software quality assurance and represents the ultimate review of specification, design and coding. Testing is the exposure of the system to trail input to see whether it produces the correct output.

6.4.1 Testing objective:

It includes the following:

• Test activities are determined and the test is selected

• The test is conducted and test results are compared with the expected results

• Various types of testing are conducted

6.4.2 Testing methods:

Testing is a process of executing a program finds out of errors. If testing is conducted successfully, it will uncover all the errors in the software.

- **White Box Testing:**
  It is test case design method that uses the control structures of the procedural design to derive test cases. Using this testing s/w engineer can derive the following test cases. Exercise all the logical decisions on either true or false sides. Execute all loops at their boundaries and within their operational boundaries. Exercise the internal data structures to assure their internal validity.
- **Black box testing:**
  It is a test case design method used on the functional requirements of the software. It will help an s/w egg. To derive sets of input conditions that will exercise all the functional requirements of the program. It attempts to find errors in the following categories: incorrect functions, missing errors, performance errors.

6.4.3 Test approaches:

Testing can be done in two ways: 1. Bottom up approach 2. Top down approach

a. **Bottom Up Approach:**
Begin with the terminal nodes of the hierarchy. A driver module is produced for every module. The next module to be tested is any module whose subordinate module has been tested.

b. **Top Down Approach:**
Begin with the top module in the execution hierarchy. Sub modules are produced, and some may require multiple versions. Stubs are more often complicated then they first appear. The next module to be tested is any module with at least one previously tested super ordinate module.

6.4.4 Types of Testing:

- **Unit Testing:**
  Unit testing is essentially for the verification of the code produced during the Coding phase and the goal is to test the internal logic of the program. The purpose is to discover Discrepancies between the modules interface specification and its actual behavior. It allows multiple testing at a time.
- **Integration Testing:**
  Integration testing is the process of combining and testing multiple Components together. The primary objective of integration testing is to discover errors in the interfaces between the components.
- **System Testing:**
  System testing verifies the entire product, after having integrated all software and hardware components and validates it according to the original Project requirements. By having a fully integrated system a tester can evaluate many attributes that cannot accessed.
- **Regression Testing:**
  This consists of reusing-a subset of previously executed tests on new versions of the applications. The goal is to ensure that feature worked on the previous versions work still expected, often, fixing a bug or by adding a new feature can break something else that once worked.

6.5 TESTING IMPLEMENTATION

Each computer, PCMCIA cards, measured received 802.11b wireless signal strength, smoothed them via moving average, and then indexed them by neighbor MAC address (ID). Hosts then mapped each measurement to respective link-reliability figures, storing them in their dynamic link-reliability arrays. Mapping calculations were based upon actual field tests of signal strength host-to-host in various positions and angular orientation with respect to each other. Ethernet signal strength measurements obtained corresponded well to those found in other work utilizing the same measurements to facilitate robot locations [13]. Utilizing this data, each host calculated its own connectivity to the rest of the MoGtestbed. Hosts dynamically exchanged calculated connectivities with neighbors via short UDP/IP packets, allowing connectivity tables, indexed by neighbor ID, and sorted max to min, to facilitate both cluster formation and ReD'scheckpointing arrangement decisions. Performance data for ReD versus RCA, were obtained with received signal strength being found to vary both positionally, and temporally.

6.5.1 Evaluation Results:

Over 1,600, 240 hour, simulations, with 32 hosts, walkingrandomly (0.5 m/s) in a 100-m square area, wereconducted, to obtain statistically broad and valid data.Simulation utilized the BigRed computer cluster (consistingof 180 nodes), located in the Center for Advanced ComputerStudies, at the University of Louisiana at Lafayette. First, theperformance of ReD was compared to that of RCA in aseries of simulations under the identically configuredconditions. Simulation data clearly showed ReD's significantimprovement over RCA. Subsequently, we verifiedReD's performance superiority over RCA through ourworking testbed. Finally, we hypothesized an intuitiveand practical stabilization mechanism for ReD, utilizingpairing reliability gain threshold.

OUTPUT SCREENS

Server:

Fig 7.1 Server for the checkpoint arrangement

Source 1:

Source 1 Connection:

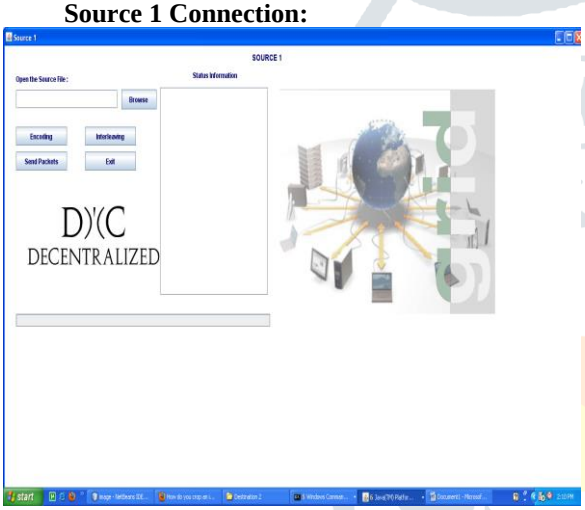

Fig 7.2 Source1 connection

Destination 1:

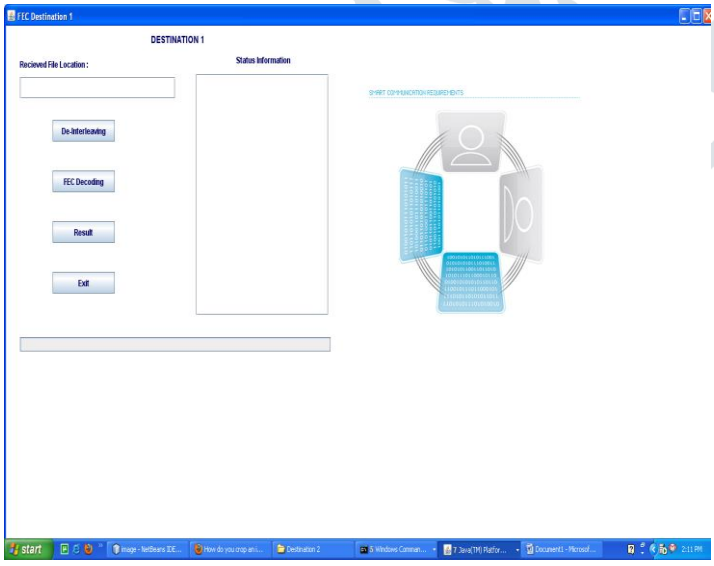

Fig 7.3 Destination1 connection

Source1:

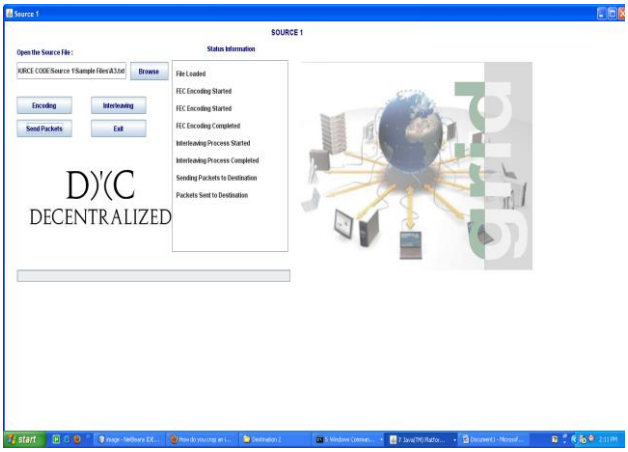

Fig 7.4 Encoding and sending packets in Source1

Server:

Fig 7.5 Server information

Destination1:

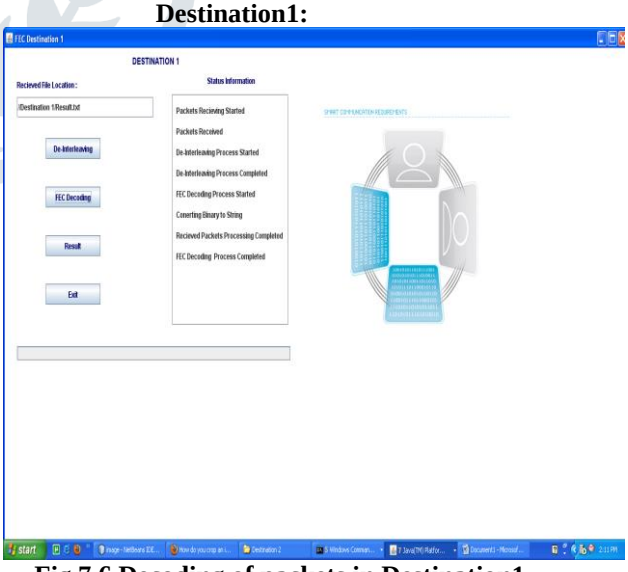

Fig 7.6 Decoding of packets in Destination1

Result:

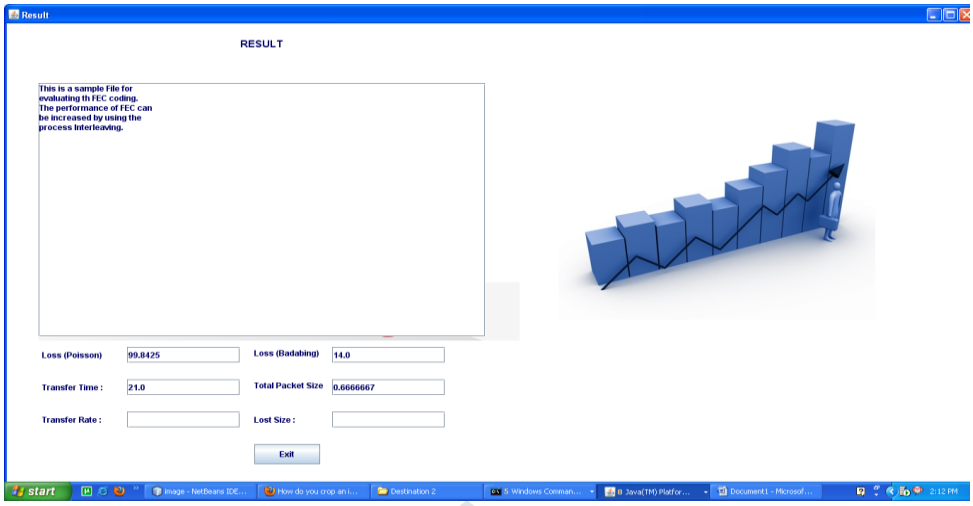

Fig 7.7 Evaluation results

CONCLUSION & FUTURE WORK

Nodal mobility in a large MoG may render a MHparticipating in one job execution, unreachable from theremaining MHs occasionally, calling for efficient checkpointingin support of long job execution. As earlierproposed checkpointing approaches cannot be applieddirectly to MoGs and are not QoS-aware, we have dealtwith QoS-aware checkpointing and recovery specifically forMoGs, with this paper focusing solely on checkpointingarrangement. It has been demonstrated via simulation andactual testbed studies, that ReD achieves significantreliability gains by quickly and efficiently determiningcheckpointing arrangements for most MHs in a MoG. Because ReD was tailoredfor a relatively unreliable wireless mobile environment, itsdesign achieves its checkpoint arrangement functions in alightweight, distributed manner, while maintaining bothlow memory and transmission energy footprints. In the proposed system the packet transfer time and its efficiency is also calculated.

The future work has marked implications for resource scheduling, checkpoint interval control, and application QoS level negotiation. It fills a component of the everdeveloping field of MoG middleware, by proposing and demonstrating how QoS-aware functionality can be practicallyand efficiently added.

REFERENCES

[1] SUN Microsystems, “Sun Grid Compute Utility,” http://www.sun.com/service/sungrid, 2006.
[2] Hewlett-Packard Development Company, L.P., “Grid-Computing-Extending Grid Computing,” http://www.1.ibm.com/grid/about_ grid/what_is.shtml, Jan. 2007.
[3] S. Wesner et al., “Mobile Collaborative Business Grids—A Short Overview of the Akogrimo Project,” white paper, Akogrimo Consortium, 2006.
[4] Computerworld, “HP Promises Global Wireless for Notebook PCs,” http://www.computerworld.com/mobiletopics/mobile/story/0,10801,110218,00.html?source=NLT_AM&nid=110218, Apr. 2006.
[5] J. Long, W. Fuchs, and J. Abraham, “Compiler-Assisted Static Checkpoint Insertion,” Proc. Symp. Fault-Tolerant Computing, pp. 58-65, July 2009.
[6] K. Ssu, B. Yao, and W. Fuchs, “An Adaptive Checkpointing Protocol to Bound Recovery Time with Message Logging,” Proc. 18th Symp. Reliable Distributed Systems, pp. 244-252, Oct. 2006.
[7] S. Bandyopadhyay and E. Coyle, “An Energy Efficient HierarchicalClustering Algorithm for Wireless Sensor Networks,” Proc. IEEE INFOCOM, pp. 1713-1723, Mar./Apr. 2008.
[8] T. Kanungo et al., “An Efficient K-Means Clustering Algorithm: Analysis and Implementation,” IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 24, no. 7, pp. 881-892, July 2008.
[9] T. Park, I. Byun, H. Kim, and H. Yeom, “The Performance of Checkpointing and Replication Schemes for Fault Tolerant Mobile Agent Systems,” Proc. 21st IEEE Symp. Reliable Distributed Systems, pp. 256-261, Oct. 2006.
[10] H. Higaki and M. Takizawa, “Checkpoint-Recovery Protocol for Reliable Mobile Systems,” Proc. 17th IEEE Symp. Reliable Distributed Systems, pp. 93-99, Oct. 2007.
[11] C. Ou, K. Ssu, and H. Jiau, “Connecting Network Partitions with Location-Assisted Forwarding Nodes in Mobile Ad Hoc Environments,” Proc. 10th IEEE Pacific Rim Int’l Symp. Dependable Computing, pp. 239-247, Mar. 2004.
[12] K. Ssu et al., “Adaptive Checkpointing with Storage Management for Mobile Environments,” IEEE Trans. Reliability, vol. 48, no. 4, pp. 315-324, Dec. 2006.
[13] G. Cao and M. Singhal, “Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing Systems,” IEEE Trans. Parallel and Distributed Systems, vol. 12, no. 2, pp. 157-172, Feb. 2005.
[14] A. Acharya and B. Badrinath, “Checkpointing DistributedApplications on Mobile Computers,” Proc. Third Int’l Conf. Parallel and Distributed Information Systems, pp. 73-80, 2007.
[15] R. Prakash and M. Singhal, “Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems,” IEEE Trans. Parallel and Distributed Systems, vol. 7, no. 10, pp. 1035-1048, Oct. 2006.
[16] N. Neves and W. Fuchs, “Adaptive Recovery for Mobile Environments,” Comm. ACM, vol. 40, no. 1, pp. 68-84, 2007.