Indian Journal of Science and Technology, Vol 10(28), DOI: 10.17485/ijst/2017/v10i28/103202, July 2017
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Optimized Application Level Checkpoint Based Load Sharing Model for Heterogeneous Mobile Grid Computing

Imran Rafique[1]*, Hina Gul[2], Salman Rafique[3], Syed Asad Raza Kazmi[1], Awais Qasim[1] and Ilyas Fakhir[1]

[1]Department of Computer Science, GC University, Lahore – 54000, Pakistan; imran.rafique@gcu.edu.pk, arkazmi@gcu.edu.pk, awais@gcu.edu.pk, fakhir@gcu.edu.pk
[2]Department of Computer Science, Kinnaird College for Women, Lahore– 54000, Pakistan; hina.gul@kinnaird.edu.pk
[3]Department of Computer of Science and Engineering, University of Engineering and Technology, Lahore – 54890, Pakistan; 2013mscs24@student.uet.edu.pk

Abstract

Objectives: Recent technical advances have fueled the popularity of mobile grid computing. Mobile devices such as cellular phones and PDAs are becoming more common due to the diminution in their size and increase of computational power. In addition, wireless networks are also beginning to fill the environment. With these advances, mobile devices are becoming available to act as service providers in Grid. But the mobile environment presents a number of challenges. **Analysis:** The range of mobile execution platforms now available which introduces the problem of heterogeneity. Heavy weight checkpoints also provide hindrance to achieve this integration. At present, Grid Computing standards, neither state any load sharing architecture and model that integrates mobile devices in Grid computing nor does it provide any policy that hides heterogeneity and overcome memory limitations of mobile devices thus it is still an open research problem. **Findings:** Mobile Grid computing solutions must be developed that are lightweight, independent of specific platform and a load sharing model for mobile grid computing that distributes computational tasks on heterogeneous mobile devices. Our simulation results show the effectiveness of data optimization techniques for mobile devices, interoperability and proxy performance in heterogeneous mobile environment. **Novelty:** We propose a novel layered architecture that adjusts the data size of checkpoints at the minimum possible level and a load sharing Mobile Proxy algorithm.

Keywords: Broker, Checkpointing, Control Flow Graph, Data Liveliness, Heterogeneity, Interoperability, Proxy, Web Service

1. Introduction

1.1 Grid Computing

Grid computing has become a hot topic in the IT industry, it provides a bright future for mega processing problems which are harder to accomplish through clusters and supercomputers. It influences the existing IT infrastructure to optimize computer resources, manages the data and computing workloads. It is because of its cost effectiveness, ability to solve large computational problems and limitation to current system used for solving large computational problems, that grid computing has gained renewed interest.

1.2 Mobile Computing

Other archetype of traditional distributing computing that is to be concerned with this research is Mobile computing. It considers mobility, wireless communication, and portability. Mobile handheld devices (e.g. Smart

*Author for correspondence

phones and Personal Digital Assistants) are enriching our daily lives and are playing vital roles in personal and business productivity because costs are decreasing and functionalities in small-sized chips are increasing day by day. The latest progress in wireless communication technology and moveable mobile equipments enable a much number of people to be eligible to access information services through a shared network transportation system (e.g., Internet) by using their mobile devices, irrespective of physical location. Moreover, Mobile environment is extremely heterogeneous. An overview of hardware, operating system, and mobile device execution platform heterogeneity in handheld devices is mentioned in Table 1.

Details of technical specifications of these phones presented in Table 1 indicate that all are different from each other. Similarly, all devices also vary in memory size of RAM. Symbian based devices have more RAM than Windows based devices. Same is the scenario with execution platforms. Windows based devices support .Net Compact framework, while J2ME works well with Symbian OS.

Although interoperability of mobile device execution platforms is not a particularly central issue in conventional distributed computing, in mobile computing, it is measured significant because hiding execution platform heterogeneity may improve mobile device portability by broadening the interoperability[1]. Thus the mobile devices can be an active participant of grid computing. Programming libraries in mobile devices are still equipped with much functionality and from these can be benefited. It has been just recently given attention to integrate these two emerging techniques of mobile and grid computing. The study of Grid computing literature reveals a lack of mature standard definitions and mechanisms to address these requirements, leaving the smooth integration and collaboration of these two technologies as a challenge. Thus, in order to provide load sharing and interoperability in an efficient manner, it is required to hide heterogeneity and complexity of collaboration.

1.3 Research Objective

The overall goal of this research is to analyze the different Application Level Checkpointing mechanisms and Middleware proposed for Grid Computing and suggest an efficient load-distribution model for mobile grid computing which will be independent of execution platform. We extend the idea of a compiler based optimization techniques to adjust the data size of checkpoints to such an extent that can reside on mobile devices and thus mobile devices can actively participate in computational tasks. Our research question is:

"How to provide an application level Checkpointing for mobile computing environment that integrates mobile devices in Grid and hides platform heterogeneity"

2. Preliminaries

Different solutions have been proposed to overcome heterogeneity in grid computing like middleware, web services based frameworks, and Application Level Checkpointing[2] etc.

Middleware implementations cannot reside on mobile devices due to memory limitations. Moreover, Literature shows that the authors have identified basic requirements that are needed by mobile devices to be an active participant of Grid computing. These factors are context awareness, dynamic environment, resource allocation, and mobile device execution platforms. Four middleware solutions proposed for Grid computing are Legion[2], Globus[2], Gridbus[2], and Unicore[2]. But none of these have support for context awareness and for dynamic environ-

Table 1. Heterogeneity in mobile computing

	NOKIA 3650	**NOKIA6630**	**NOKIA N90**	**IPAQ H3650**	**DELLL AXIM**
OS	Symbian OS 6.1	Symbian OS 8.0a	Symbian OS 8.1	Windows Mobile	Windows Mobile
Processor	104 MHz	220 MHz	220 MHz	206 MHz	624 MHz
RAM	1.9 MB free of 4 MB	7 MB free of 10 MB	21 MB free of 48 MB	32 MB	64 MB
Execution Platform	JAVA	JAVA	JAVA	.Net CF	.Net CF

Table 2. Support for mobile devices in grid middleware

Middleware	Description	Support for Handheld device requirements
Globus	Globus is based on a bottom-up paradigm, and provides a set of tools for distributed applications in the form of a toolkit.	• No support for context awareness, dynamic environment, and mobile device execution platform. • Has support for resource allocation.
Gridbus	It is an open source grid software Many middleware, Globus, Unicore, Alchemi, and others are used in Gridbus.	• No support for context awareness, dynamic environment. • Has support for resource allocation, and mobile device execution platform.
Legion	UNICORE (UNIform Interface to COmputer REsources) middleware Provides a user interface to integrate grid computing resources.	• No support for context awareness, dynamic environment, and mobile device execution platform. • Has support for resource allocation.
Unicore	Legion is based on top-down approach; services are treated as objects and based on communication paradigms.	• No support for context awareness, dynamic environment, and mobile device execution platform. • Has support for resource allocation.

Table 3. Checkpointing systems and techniques

Technique	Strengths	Limitations
Condor	First system for migrating threads in distributed environment	Uses system level Checkpointing. Transfers huge bulks of data. Does not hide heterogeneity.
Migthread	Migthread distributes threads to different machines or file systems. User level stack and heap management Data conversion only at receiver side.	Limited to C Programs. No Mechanisms to reduce data size.
Efficient Checkpointing for Hetero-geneous Collaborative Environment	Migrates process to heterogeneous machines	Uses PVM for process migration. Only work for C, FORTRON Encode and Decode data types in machine independent formats thus execution complexity class varies.
Mobile MPI Programs in Computational Grids	Based on portable application level Checkpointing. Truly Heterogeneous for MPI Programs.	MPI dependent. Uses a Pre-compiler for data conversions. Checkpoint data can be up to 1GB Over-decomposition sometimes creates a significant bottleneck.
Portable Checkpointing for BSP App.	Support execution of BSP parallel applications on heterogeneous, shared workstations. Data types Conversion only at receiver side.	Dependent on thread communication libraries. Still restarting application from checkpoint creates overhead. Works only for C++ Programs. No mechanisms for reducing the size of the checkpoint file.
Automated Application level Check-pointing based on Live variable analysis for MPI Programs	Enhances idea of Checkpointing techniques by using data analysis. The data size is reduced.	Still dependent on thread communication libraries. Data type conversion both at the sender and receiver sides. Application-level coordinated non-blocking Checkpointing. Can be used for C/MPI Programs

ment and only Gridbus support mobile device execution platform that is a .Net Compact Framework. Middleware support for these factors is summarized in Table 2.

Second technique for hiding heterogeneity is Checkpointing. Various authors worked on this concept[3-9]. All work relevant to Checkpointing is described in Table 3 with their strengths and limitations.

A third technique for hiding heterogeneity is web service based frameworks[10]. Much work is being done to expand web services technology to Mobile computing domain. By doing these limitations of physical locations will be diminished due to the mobility feature of the Mobile computing environment. Web services can be an ideal solution that provides a connection between mobile and Grid computing and hides heterogeneity too because web services are strongly interoperable. Both computing domains will be equally benefited as a result of this integration. In spite of a lot of enhancement in the Mobile computing domain, this integration might cause certain performance overheads. There are two reasons for this performance degradation.

- Resources are consumed by encoding and decoding of messages.
- The Message size is increased due to textual format. It contains data and descriptive tags and in the case of redundant data structures such as arrays, it can be up to many folds and it will create problems in wireless domains because of limited bandwidth.

3. System Architecture

3.1 Elaboration of Proposed Design

From the literature investigation in the previous section, it is concluded that Application level Checkpointing must be independent of communication libraries and execution platforms, data must be adjusted to the memory size of handheld devices and there is a need to hide the heterogeneity at:

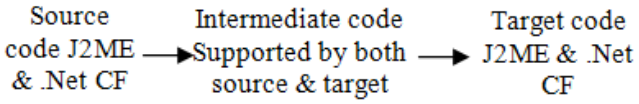
Execution Level

Interoperability of execution platforms

Communication Level

Common set of Protocols

Checkpoint data in an independent format



4. Proposed Architecture

Since mobile devices have certain constraints, due to which integrating mobile devices into the grid will reduce efficiency and may create bottlenecks and grid environment may become engaged in handling these issues rather than involved in computational tasks. So there should be varied approach that not only integrate mobile devices into the grid as service providers, but also overruns these devices from some of the constraints like heterogeneity and memory overflow. Keeping this in view, we propose layered design based on proxy. This proxy acts as a gateway between mobile devices and grid environment.

4.1 Why to Add Proxy?

There is a radical improvement in wireless network connections. But there are certain barriers that degrade performance of wireless network connections to equalize with that of wired systems.

1. 3G technology provides maximum bandwidth 300~500kbps for downloading and 56~90kbps for uploading, and these cannot cope with the bandwidth requirements of wired computing domains.
2. Degradation factors like buildings and landmarks in wireless connections based radio waves.
3. Mobile devices have smaller memories and slower processors.

Eventually, all these factors require that some processing must be done on all data before transferred to mobile devices using wireless networks.

4.1.1 Layer 1: Service Request and Service Publicizing

At proxy various components of Grid Middleware and data optimization are deployed. It has communication interfaces for wired Grid clients and as well as for mobile devices too. Communication environment is a merger of Grid protocols like GRAM, GSI, MDS and of web service

protocols. The main purpose of this layer is to publish services, request and discovery of services, and getting request data from client. It is performed in the following manner.

4.1.1.1 Service Publicizes

This process enables all clients and users to publish and view details of services and resources and other metadata information[11]. A registry (online database) is maintained for publication, querying a service, and for information of resources as UDDI registry. When any device wants to publish its services, it performs publish operation on registry and sends its web link and resource details as parameters. After this operation, service is registered. Then other authorized clients can view this list. Similarly, the devices can disconnect by invoking de-publish method.

4.1.1.2 Request and Discovery of Services

Grid clients submit jobs called requests by using GRAM protocol. A single request may consist of several parallel executing operations[12]. Thus a single operation is assigned to any single devices. Devices or services are searched by discovery module.

4.1.1.3 Broker

Broker creates a sort of virtual environment on the behalf of the requesting client. After finding available service, the request data is transferred from client to broker. It performs scheduling of jobs. It sends data of scheduled job to controller for the code optimization module to reduce data size in order to cope with memory limitations of mobile devices.

Layer Output: Arrival of service data at proxy and delivered to the controller.

4.1.2 Layer 2: Data Analysis and Construction of Control Flow Graph

After getting service data next step is to apply some optimization techniques to reduce checkpoint data and to overcome limitations of web services. Various compilers based optimization techniques[13] are in existence, but these techniques are not in practice for mobile devices. The process of data analysis of optimization is called static analysis. Static analysis is divided into two main categories. First one is construction of control flow graph. The second is a data-flow. In Control-flow analysis, hierarchical flow of program control is examined within each fragment of program flow chart. The Data-flow analysis identifies live and dead data sets for each fragment. The broker sends data to the controller. Controller divides data into fragments. Fragmentation is not a trivial case. There can be multiple possibilities for one program to fragment it. It depends on the following possibilities:

- Locality of two consecutive fragment points is considered.
- Data size of fragment.

Each fragment is assigned a unique id. To assign this id all instructions are converted into three address instructions the key instructions are identified that transfer control to another location. These can be:

- Function Definitions, Start of Loop, and Switch cases

Fragment id is added prior to the key instruction and instructions following this are added in the same fragment till the arrival of the next key instructions. When next key instruction is identified, fragment id is incremented. After the dividing program into fragments a control flow graph is constructed that depicts the transfer of control from main fragment to the subsequent fragments.

Layer Output: Control Flow Graph.

4.1.3 Layer 3: Data Liveliness Analysis

In this step live data variables are calculated in all blocks of CFG. The data-flow analysis identifies live and dead data sets for each fragment. In live and dead variable analysis[14,15] program CFG is traversed. For each fragment, dead variables and live variable are identified. Any variable whose values is required in the processing of successor fragments are live variables and all those variables whose values are not required in successor fragments are dead variables. Then dead variable sets are discarded and only live data sets are transferred to handheld devices. Equations used for this process are given below.

$$in[B] = use[B] \cup (out[B[- def[B])$$

$$out[B] = \cup_{s \text{ a successor of } B} in[B]$$

To address memory constraint data blocks is analyzed to reduce data size so that light weight checkpoint is sent[16]. Thus, the idea of checking liveliness of data has extended to mobile devices.

Layer Output: Live data and dead data sets of all fragments.

4.1.4 Layer 4: Request Distribution and Hiding Heterogeneity

Service requests consist of independent processes and testing concurrent systems[17] is a critical task. Furthermore, these processes may consist of independent and dependent threads that may execute parallel or sequentially. Processes are the programs written in any language. At upper layers, independent and dependent threads in a process have been identified by constructing a control flow graph and thread data has also been optimized by liveliness analysis. Now, at this layer, techniques for hiding heterogeneity are provided. Since checkpoint data is sent and received by using XML format and SOAP protocol of web services. Thus, checkpoint data is encoded in XML binaries and transferred to the resource provider device[18]. This destination device can be equipped with Windows CE or Windows Mobile supporting .NET Compact Framework or it can be equipped with Symbian OS with J2ME execution Platform. Figure 1 shows in XML binaries are decoded according to the underlying platform. Flow diagram of Proposed Architecture is given in Figure 1. For this purpose, device characteristics like Opesys, ExePForm, and Arche are retrieved. And XML binaries are decoded according to the values of these features. For example, if device characteristics Opesys = Windows CE and ExePForm = .NET CF and Arche = "", then binary data will be accorded the format of .NET CF supportable by Windows CE. And the data is processed and results are calculated and sent back to the controller.

Pseudo code is as shown below and this procedure is to be executed on mobile devices:

Start procedure

if(Opesys = Windows CE && ExePForm = .NET CF && Arche="")

then

decode and execute data to CE.NET;

else if(Opesys = Windows Mobile && ExePForm = .NET CF && Arche="")

then

decode and execute data to Mobile.NET;

else if(Opesys = Symbian && ExePForm = J2ME && Arche="")

then

decode and execute data to SymbianJava;

else

break;

end

Layer Output: Hiding heterogeneity of checkpoint data.

4.1.5 Layer 5: Monitoring and Message Handling

After starting execution of threads, controller monitors execution and handles messages, and results are sent back to the controller. After getting results, controller combines these and sends to broker.

Step # 01: *Controller* class gets separate blocks of control flow graph that are arranged in an array data structure

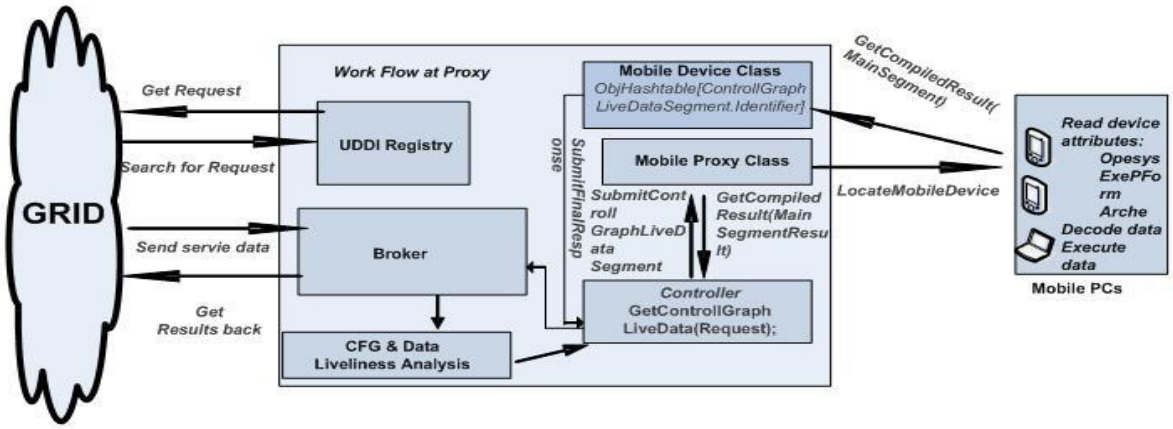


Figure 1. Proposed architecture.

namely *ControllGraphLiveDataArrayList*. For each block, controller searches for free mobile devices and on success, it assigns a separate block to separate device and wait for results.

Step # 02: In *MobileProxy* class, every mobile device that is assigned to perform any task by Controller Class gets the data of request and initiate threads and starts processing in a method namely *SubmitControllGraphLiveDataSegment(ControllGraphLiveDataSegment)*.

Step # 03: Mobile Device class gets results of each segment from all devices and maintains a hash table of results, on getting results from all devices it sends all results back to Controller class.

The controller checks status of devices, whether these are still living and executing operations. If any device disconnects, controller at once search for any other device offering services, otherwise it will wait for devices busy in execution to be freed.

Layer Output: Status of devices is checked.

An algorithm that sums up whole steps as mentioned in all levels and describes overall working of Mobile Proxy is as shown below and this procedure is to be executed on Mobile devices:

Optimized Control Flow Graph (Application level Checkpointing) based MobileProxy Algorithm for the Proposed System

```
classController
{
staticvoid MainRequestController(Request)
{
ControllGraphLiveDataArrayList = GetControllGraphLiveData(Request);
int Counter=0;
while (Counter <= ControllGraphLiveDataArrayList.Length)
{
MobileProxy.SubmitControllGraphLiveDataSegment(ControllGraphLiveDataArrayList[Counter]);
Counter++;
}
MainSegmentResult = Compute(ControllGraphLiveDataArrayList.Data);
CompiledResponse = MobileProxy.GetCompiledResult(MainSegmentResult);
// To Orignal Source
SubmitFinalResponse(CompiledResponse);
}
}
classMobileProxy
{
staticvoid SubmitControllGraphLiveDataSegment(ControllGraphLiveDataSegment)
{
MobileDevice=LocateMobileDevice(ControllGraphLiveDataSegment.Specs);
Thread ProcessSegment =newThread(MobileDevice.ProcessControllGraphLiveDataSegment);
ProcessSegment.Start();
}
static Result GetCompiledResult(MainSegment)
{
Result = MobileDevice.GetCompiledResult(MainSegment);
return Result;
}
}
classMobileDevice
{
staticHashtable ObjHashtable=newHashtable();
staticvoid ProcessControllGraphLiveDataSegment()
{
ControllGraphLiveDataArrayList = GetControllGraphLiveData(ControllGraphLiveDataSegment);
MainRequestController();
if(ControllGraphLiveDataArrayList.Length>0)
Controller.MainRequestController(ControllGraphLiveDataArrayList);
Result=ComputeResult(ControllGraphLiveDataSegment);
ObjHashtable[ControllGraphLiveDataSegment.Identifier]=Result;
}
static Result GetCompiledResult(MainSegment)
{
while(WaitforAllControllGraphLiveDataSegmentThreads());
Return (ComputeFinalResult(ObjHashtable));
}
}
```

5. System Evaluation

In this section, the simulation setup and performance analysis of the proposed design is described. Different issues that may have a positive and negative impact on the overall performance of the proposed system are

considered and evaluated. The proposed design is to be evaluated for three main factors that are:

1. Reduction in data size as a result of optimization techniques
2. Mobile Proxy overhead in terms of load, delay, throughput, and proxy CPU utilization of mobile devices.
3. Interoperability overhead while distributing .net CF code to J2ME and while running on the same platform.

Each of the above mentioned three performance measures is discussed in the subsequent sections, one by one along with their simulation setups and evaluated results.

5.1 Analysis of Code Optimization Technique

The data optimization procedure is to be executed on Mobile Proxy for reduction in data size that is to be transferred on mobile devices. It is estimated that as a result of this procedure data size is reduced so the data overhead will be low, but at the same time some time factor will be involved as compared to a traditional Grid environment where there is no need for making process data light weight.

5.1.1 Data Size Overhead

Original data size that is transferred is comprised of following parts:

$D_s = D_{env} + D_{tdata} + N(D_{des}) + D_{att}$

D_{env} = Envelop size

D_{tdata} = Total data

D_{des} = Data description

D_{att} = Attributes of underlying platform

N = Total no. of operations

But due to code optimization, dead data is eliminated, thus total data size is reduced and data overhead is less:

$D_s = D_{env} + (D_{tdata} - D_{dead}) + N(D_{des}) + D_{att}$

D_{env} = Envelop size

D_{tdata} = Total data

D_{des} = Data description

D_{att} = Attributes of underlying platform

N = Total no. of operations

D_{dead} = Dead data variables

5.1.2 Simulation Results for Code Optimization

Simulations for code optimization are carried in Project Analyzer and Code Visual to Flowchart tools by using five different real time applications that involve simple arithmetic, control transfer, audio data, database access, and live streaming video applications.

Data optimization involves three steps:

1. Construction of Control Flow Graph
2. Identification of independent and dependent threads
3. Identification of Live data

Construction of Control Flow Graph

For this purpose CFGs are constructed for source codes of all applications by using Code Visual to Flowchart tool.

Identification of threads

By using Project Analyzer tool independent and dependent threads or blocks of CFG are identified.

Identification of Live data

In this step live data variables are calculated in all blocks of CFG. Equations used for this process are given in layer 3. For the simulations of this research, after the construction of Control Flow Graph and identification of independent and dependent blocks, we selected two blocks from all code samples in which one thread is dependent on its predecessor and gets data from it. Each block along with their live and dead code is given below. From the entire given sample codes, number of live and dead variables are calculated and the percentage of data reduction is given in Table 4 and Figure 2 and it is clearly depicted that there is a significant reduction of data size of the message.

Table 4. Code optimization results

Name of application	Lines of code	Data size	Opt. Data size	Reduction in size	% age of live data
Arithmetic	5	4	2	50%	50%
Control transfer	10	6	5	17%	83%
Audio Data	15	4	1	75%	25%
Database access	20	15	3	80%	205
Live streaming Video Data	25	17	8	53%	47%

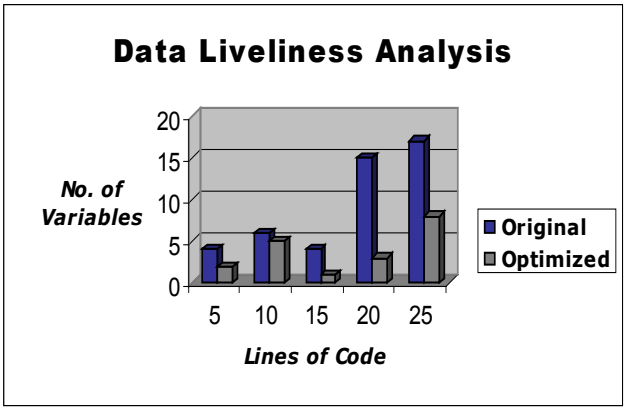


Figure 2. Graphical representation of optimization.

5.2 Analysis of Proxy Performance

A proxy, in this research, is a server or workstation or router as wireless access point which provides additional data and code functionality and it is deployed at the border of wireless network and internet, and is used to perform data liveliness analysis and encoding and decoding of data into XML binaries for all requests that arrives from Grid environment. A control flow graph of source code is constructed at the proxy and live and dead code is identified by using code optimization algorithm. After that dead code is eliminated and the index is attached with all live variables and all live data of independent threads is encoded in XML format and distributed in a mobile environment. Similarly, all threads return their results at proxy in XML format and proxy decoded it to get the original data format. By performing data analysis and code optimization, there is a significant reduction in message size, however, some delay factor is estimated in doing all this processing as compared to any distributed environment (wired or wireless) where proxy is not implemented.

5.2.1 Time Complexity Evaluation

Without proxy, there are only three factors that add value to time complexity.

$T = T_{com} + T_{res} + T_{mrg}$

T_{com} = Communication time

T_{res} = Response time

T_{mrg} = Combining results time

While, with the addition of proxy data analysis and code optimization also adds values to determine time complexity. In this way total time of processing in Mobile Grid environment is also increased.

$T = T_{com} + T_{res} + T_{mrg} + T_{cfg} + T_{dla} + T_{end}$

T_{com} =Communication time

T_{res} =Response time

T_{mrg} =Combining results time

T_{cfg} = CFG construction time

T_{dla} =Data liveliness analysis time

T_{end} = Time involved in Encoding and decoding into XML format

To measure delay, throughput, and load on the Mobile Grid environment with and without proxy, all simulations are carried out in Optimum Network Performance (OPNet) simulator.

Two scenarios create one of which has proxy and another is without proxy. A brief description of project scenarios is as follows:

1. An IP32-Cloud object represents a Grid environment.
2. An Access Point (Proxy) with wireless LAN configurations in Table 5 that gets a request from Grid and distribute to all mobile devices.

Table 5. WLAN settings

Characteristics	Value
Data rate	11 Mbps
Physical Characteristics	Direct Sequence
Access Point access	Enabled

Applications Config object and Profile Config object were configured for three types of mobile services with following settings in Table 6.

Table 6. Application settings

Application Processes	Value
Database Processes	Heavy
FTP Processes	Heavy
Web Processes	Heavy

Total 15 Mobiles devices all of which 60% of their CPU cycles to Grid Processes. From all these devices, 5 devices provide database services, 5 devices support FTP services, and 5 devices support web services. These devices are configured to send and receive requests for their supporting processes.

Proxy Implementation

In one scenario, proxy is also implemented between Grid environment and Access Point. This performs data analysis and code optimization for all messages of

Database, FTP and Web processes. All data from grid arrives at this proxy, after processing all data is passed to the Access Point from where it is distributed. The second scenario is without the proxy.

5.2.2 Simulation Results of Proxy Performance

Proxy Behavior

Proxy distributes load as shown in Figure 3 in both scenarios equally, but it works, in case of proxy, little bit more but does not create the delay factor.

Proxy CPU Utilization

While analyzing proxy CPU utilization in Figure 4, it is observed that proxy is actively involved in doing all processing of data analysis and code optimization.

Mobile Device Behavior

Mobile devices are also analyzed for four parameters.

1. Delay
2. Load
3. Throughput

Both scenarios are analyzed for one hour. In scenario 01 with Mobile Proxy, there is little bit delay but it can be ignored. But load on mobile devices is lower due to smaller message sizes. And throughput is same in both cases as shown in Figure 5-7.

5.3 Analysis of Interoperability

In this section, it is evaluated that executing .NET CF code on J2ME and .NET CF and then execution time are

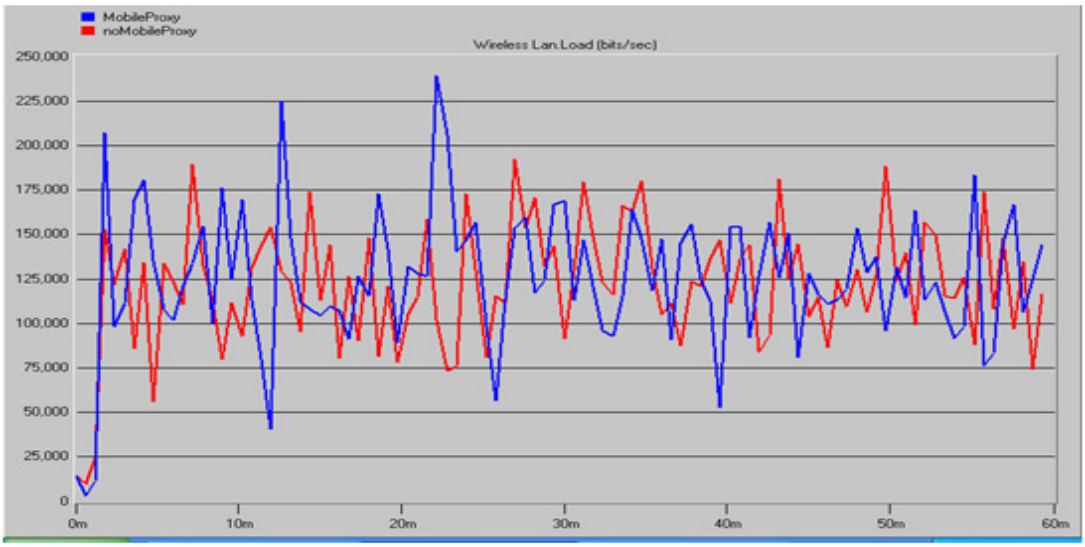


Figure 3. Proxy load.

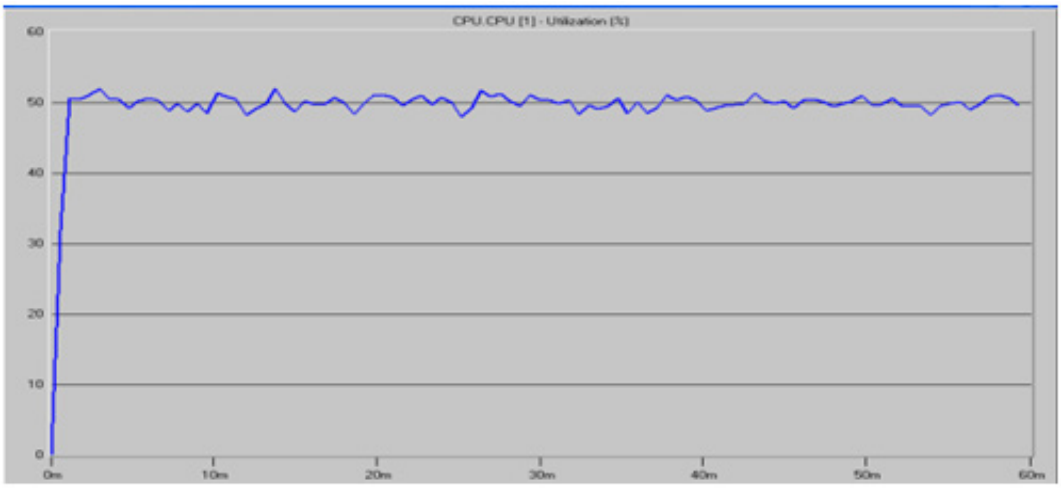


Figure 4. Proxy CPU utilization.

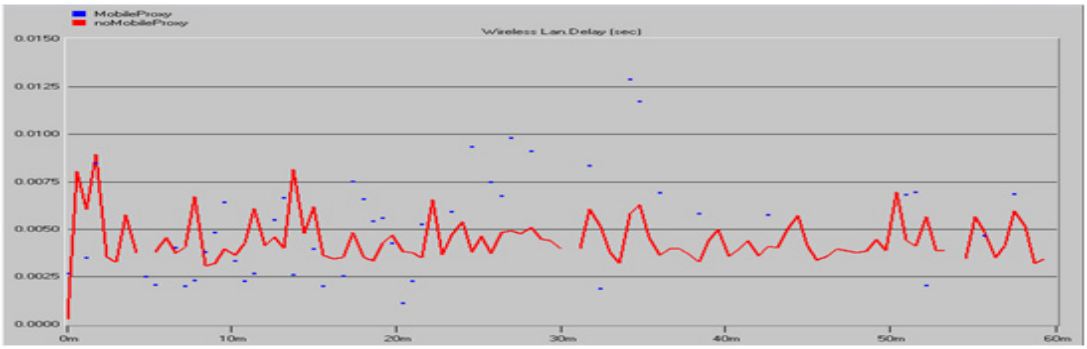


Figure 5. MobilePC delay.

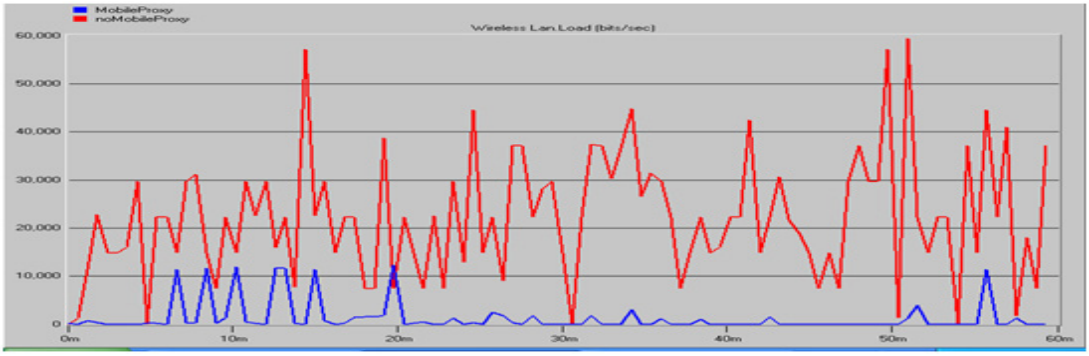


Figure 6. MobilePC load.

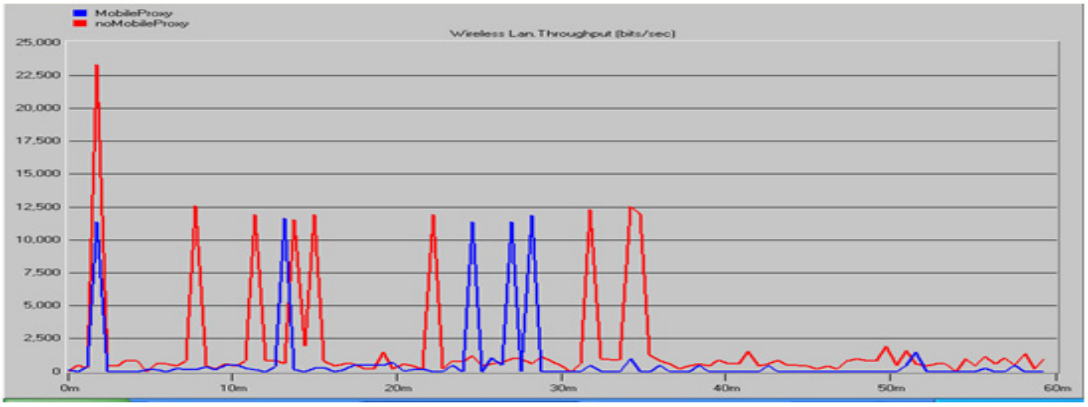


Figure 7. MobilePC throughput.

compared. For this purpose, we use micro-benchmarks. These are simple programs (such as loops) that targets sole functionality. We choose a set of micro-benchmarks for carrying out the evaluation; Devices used for these tests have the following specifications:

1. JAVA Platform
   - Symbian OS, 10 MB of Memory, and 220 MHz Processor

2. .Net CF Platform
   - T-Mobile MDA II Pocket PC

5.3.1 Simulation Results of Interoperability

Table 7. Micro-bench marks details

	Micro-benchmarks	**J2ME**	**.NET CF**
Bm1	MemReadLatency	141	122
Bm2	Method Calling	203	330
Bm3	Apawm Thread	31	61

Bm4	StringCompare	328	217
Bm5	CopyArray	328	389
Bm6	InitArray	250	166
Bm7	SumArray	16	15

Table 7 shows execution of all micro-benchmarks, it is significantly from the results that interoperability has no negative impact because some micro-benchmarks works better on .Net CF and some works well on J2ME. Slighter variations are due to the result of internal algorithms of both execution platforms. Same results are represented in the graph Figure 8.

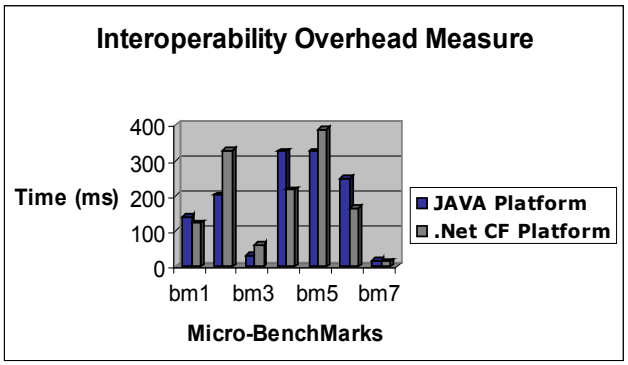


Figure 8. Performance of interoperability.

6. Conclusion

In this research, we have shown a novel light weight checkpoint data distribution design and algorithm which also hides heterogeneity is proposed and its performance has been evaluated. Furthermore, the concept of application level checkpointing is enhanced for mobile devices in traditional Grid Computing solutions. While an important factor, an interoperability of .Net Compact Framework and J2ME has been fully addressed in this research.

The Simulation results show that interoperability is possible and source code written on one platform can be executed on the destination platform with ignorable execution time differences. Proposed design overruns previous solutions in many perspectives such that optimization techniques reduce data size significantly so that checkpoints can reside on mobile devices without creating bottlenecks such as memory overflow and degradation in performance. Performance of proxy introduced in the proposed algorithm, is tested through simulations and results show that no significant degradation is observed in a wireless environment as compared to traditional infrastructures that are without the proxy. It is also shown from the simulation results that mobile devices can actively participate in Grid Computing if some preprocessing is done at checkpoint data on the proxy. Moreover the algorithm proposed is independent of process migration libraries.

7. Future Directions

Many enhancements can be proposed in respect to future work of this research. But the most significant is that there must be separate client and server side toolkits for the proposed layered architecture.

8. References

1. Antonio P.J, Balaji P. A data-oriented profiler to assist in data partitioning and distribution for heterogeneous memory in HPC, Parallel Computing. 2016 Jan; 51:46–55. Crossref.
2. Greg B. Application-level checkpointing for shared memory programs, ACM SIGARCH Computer Architecture News. 2004 Dec; 32(5):235–47. Crossref.
3. Nuria L. Resilient MPI applications using an application-level checkpointing framework and ULFM, The Journal of Supercomputing. 2016 Jan; p. 1–14.
4. Iván C. Achieving checkpointing global consistency through a hybrid compile time and runtime protocol, Procedia Computer Science. 2013 Dec; 18:169–78. Crossref.
5. The Physiology of the Grid. Data accessed: 02.07.2016. http://www.globus.org/alliance/publications/papers/ogsa.pdf.
6. Yunfei D, Tang Y, Xie X. A new parallel re-computing code design methodology for fast failure recovery, Computers and Electrical Engineering. 2013 May; 39(4):1095–13. Crossref.
7. Righi D.A, Rodrigo. Observing the impact of multiple metrics and runtime adaptations on BSP process rescheduling, Parallel Processing Letters. 2010 June; 20(02):123–44. Crossref.
8. Vazhkudai, Sudharshan S. Constructing collaborative desktop storage caches for large scientific datasets, ACM Transactions on Storage (TOS). 2006 Aug; 2(3):221–54. Crossref.
9. Bastian S, Brauer J, Kowalewski S. Application of static analysis for state-space reduction to the microcontroller binary code, Science of Computer Programming. 2011 Feb;76(2):100–18. Crossref.
10. Dieter F, Bussler C. The web service modeling framework WSMF, Electronic Commerce Research and Applications. 2002; 1(2):113–37. Crossref.

11. Peter M.A. Data publication with the structural biology data grid supports live analysis, Nature Communications. 2016.
12. Gang Z. Large-scale, high-resolution agricultural systems modeling using a hybrid approach combining grid computing and parallel processing, Environmental Modelling and Software. 2013 Mar; 41:231–38. Crossref.
13. Toma, Ioan. Discovery in grid and web services environments: A survey and evaluation, Multiagent and Grid Systems. 2007 Aug; 3(3):341–52. Crossref.
14. Baldwin, Douglas, Sayward F. Heuristics for determining equivalence of program mutations. Georgia Institute of Technology School of Information and Computer Science, 1979.
15. Pu liu. Mobile code enabled web and grid services, Publication Company ProQuest, 2006.
16. Kaur, Shubhinder, Kaur G. Weight based task assignment model to tolerate faults in heterogeneous distributed systems, International Journal of Computer Applications. 2015 Sep; 125(9):25–28. Crossref.
17. Fakhir, Ilyas. Concurrency in intuitionistic linear-time μ-calculus: a case study of manufacturing system, Indian Journal of Science and Technology. 2016 Feb; 9(6):1–7. Crossref.
18. Rathore, Neeraj, Chana I. Load balancing and job migration techniques in grid: A survey of recent trends, Wireless Personal Communications. 2014 Dec; 79(3):2089–125. Crossref.