View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by The Research Repository @ WVU (West Virginia University)

Graduate Theses, Dissertations, and Problem Reports

2006

Efficient simulation of communication systems on a desktop grid

Raja Sekhar Katuri
West Virginia University

Follow this and additional works at: https://researchrepository.wvu.edu/etd

Recommended Citation

Katuri, Raja Sekhar, "Efficient simulation of communication systems on a desktop grid" (2006). *Graduate Theses, Dissertations, and Problem Reports*. 2391.
https://researchrepository.wvu.edu/etd/2391

This Thesis is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Thesis has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.

Efficient Simulation of Communication Systems on a Desktop Grid

by

Raja Sekhar Katuri

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Electrical Engineering

Matthew Valenti, Ph.D., Chair
Brian Woerner, Ph.D.
Don McLaughlin

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2006

Keywords: Desktop Grid, Distributed Simulation, Global Grid Exchange

Copyright 2006 Raja Sekhar Katuri

Abstract

Efficient Simulation of Communication Systems on a Desktop Grid

by

Raja Sekhar Katuri

Simulation is an important part of the design cycle of modern communication systems. As communication systems grow more sophisticated, the computational burden of these simulations can become excessive. The need to rapidly bring systems to market generally precludes the use of a single computer, and drives a demand for parallel computation. While this demand could be satisfied by the development of dedicated infrastructure, a more efficient option is to harness the unused computational cycles of underutilized desktop computers located throughout the organization.

In this thesis, a new paradigm for parallelizing communication simulations is proposed and developed. A desktop grid is created by running a compute engine as a background job on existing computers located throughout the University. The compute engine takes advantage of unused cycles to run simulations, and reports its results back to a server. The simulation itself is developed and launched from a client machine using Matlab, an application that has widespread acceptance within the communications industry. To obviate the need for a Matlab license on every machine running the compute engine, the simulation is first compiled to stand-alone executable code, and the executable and input data files are distributed to the grid machines over the Internet. To illustrate the performance improvement, a campaign of 16 distinct simulations corresponding to the IEEE 802.11a standard is run over the grid. Each compute engine executes a single simulation corresponding to one of eight modulation and coding schemes and one of two channel models. The improvement in execution time is quantified by a tool that was developed to monitor the activity of the grid.

I dedicate this thesis to my sister K. Udaysri.

Acknowledgments

I take this opportunity to thank my advisor Dr. Valenti for all his support in the completion of my thesis. I also thank Dr. Woerner and Don McLaughlin for serving in my committee.

This work would not have been possible without the contribution of Jim O' Connor to who am i am greatly indebted. I acknowledge the support of Parabon Computation Inc. and LDCSEE systems staff in West Virginia University. I am grateful to my friends for all the help they have extended.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Objectives

The communications industry is enormous. For instance, there are nearly 2 billion wireless phones worldwide[1]. In the United States alone, there are 217 million wireless phones and over 180 thousand cellular base stations[2]. In 2005, the US wireless industry brought in over $113 billion in revenue and employed 233 thousand workers. With such high stakes, new wireless communication networks must be thoroughly tested before they are fielded.

Monte Carlo simulation has become a widely accepted critical part of the design cycle of communication systems. Often, the simulations are implemented in Matlab, which has become a "defacto" standard platform for developing communication simulations. Because of the complexity of modern communication systems, these simulations are very compute intensive. As the computational demands of most simulation campaigns cannot be met by a single computer, there is a need for parallelizing communication simulations, for instance by using grid computing. It is the goal of this thesis to develop a methodology for running Matlab-based communication simulations in parallel over a grid computer.

To obviate the need for a Matlab license on the machines being harnessed by the grid, Matlab programs are compiled into stand-alone executables (for both windows and Linux platforms). As a specific example, the grid is used to simulate the coding and modulation

[1] www.wirelessintelligence.com

[2] Cellular Telecommunications Industry Association,www.cita.org

used in the IEEE 802.11a standard. The benefit of running the simulations on a grid is improved execution time. Lengthy simulation campaigns that might take weeks or months to complete on a single machine can be finished in a matter of hours or days on the grid. The grid itself is formed from networks of ordinary PC's by harnessing their idle CPU time. The key point to be noted is that the computers used in the grid are not dedicated systems specifically to be used only for running simulations. Rather, they are existing resources that can be located in student computing laboratories, libraries, and offices throughout campus. This opportunistic use of the existing resources is made possible by using the distributed computing platform *Frontier* which is hosted by the Global Grid Exchange and developed by Parabon Computation, Inc..

One way to utilize the grid is to execute just one simulation at a time, breaking it up so that each machine on the grid is responsible for only a fraction of the required Monte Carlo trials. This is a good choice when there is only a single simulation to run and it must be completed quickly. However, it has the disadvantage that the results from the various machines must be combined in an appropriate manner, and the simulation is ultimately limited by the slowest machine. Another alternative, and the one embraced by this thesis, is to run a separate simulation on each grid node. Each simulation could correspond to a different simulation parameter, such as modulation or coding type. This is appropriate when the simulation campaign is exploring several options, or is simulating a system with many different settings. For instance, in the IEEE 802.11a standard there are 8 distinct data rate options, and so a simulation campaign that considers all of these rate options could be parallelized by simulating one data rate option on each of 8 grid nodes. If two channel types are to be considered, for instance AWGN and Rayleigh, then there will be 16 simulations in the campaign, which can be executed in parallel on 16 machines.

1.2 Thesis Outline

Chapter 2 of this thesis deals with modulation and coding techniques and presents simulation results for the coding and modulation used by the 802.11a standard. Bit Interleaved Coded Modulation (BICM) has been incorporated in these simulations, since it is used by

802.11a. The chapter continues by discussing convolutional coding and the concept of puncturing. The convolutional encoder used by the IEEE 802.11a standard [2] is presented. It also discusses the concept of *Shannon capacity* and describes Monte Carlo methods for computing the Shannon capacity of channels that use the modulation formats from the 802.11a standard. Note that these simulations presented in this chapter are performed on a single machine.

Chapter 3 introduces the concept of grid computing, a technology that enables the sharing of distributed powerful computing resources. It provides a brief history of grid computing by reviewing earlier distributed computing projects. It further discusses the various Supercomputing technologies from which grid computing has evolved. This chapter also discusses various grid models and explains the basic characteristics of a desktop grid.

Chapter 4 introduces the grid computing platform *Frontier* provided by the Global Grid Exchange and further discusses implementation issues involved in utilizing the grid for simulations. The user interface is Matlab and the Matlab programs are compiled to stand-alone executables using the Matlab compiler. The compiled executable is sent to each machine on the grid, along with files containing the simulation parameters and simulation state (sending the state allows previously run simulations to be resumed). The state of the simulation is periodically reported back to the end user's system, so that the simulation progress can be tracked and intermediate results plotted. The chapter concludes with the evaluation of the grid's performance in terms of computational throughput. Finally, chapter 5 provides conclusions and suggestions for future work.

Chapter 2

Modulation and Coding

This Chapter describes the system model and details the implementation of baseband transmission and reception, including modulation and coding. It describes the concepts of coded modulation (CM) and bit interleaved coded modulation (BICM), and discusses Monte Carlo methods for determining Shannon capacity. Finally, it concludes with simulation results showing the bit error rate (BER) of both coded and uncoded systems using parameters from the IEEE 802.11a standard.

2.1 Modulation

Digital modulation is the process in which binary information is mapped to signals by taking groups of $\mu = \log_2 M$ bits from the input sequence and selecting a symbol x_k from the signal set $\mathcal{S} = \{x_1, x_2, ..., x_M\}$. With binary phase shift keying (BPSK), $M = 2$, and the signals are one-dimensional. Quadrature phase shift keying (QPSK) and quadrature amplitude modulation (QAM) are examples of two-dimensional modulation techniques which are formed by varying the amplitude and/or phase of a signal. Two dimensional signalling is conveniently handled by the use of complex numbers, with real and imaginary parts of the signal x_k corresponding to the in-phase and quadrature parts, respectively.

The received signal in a frequency non-selective communication system can often be characterized as $y_k = h_k x_k + n$ where h_k independent and identically distributed complex fading coefficient and n is random noise, usually modelled as a complex Gaussian random vari-

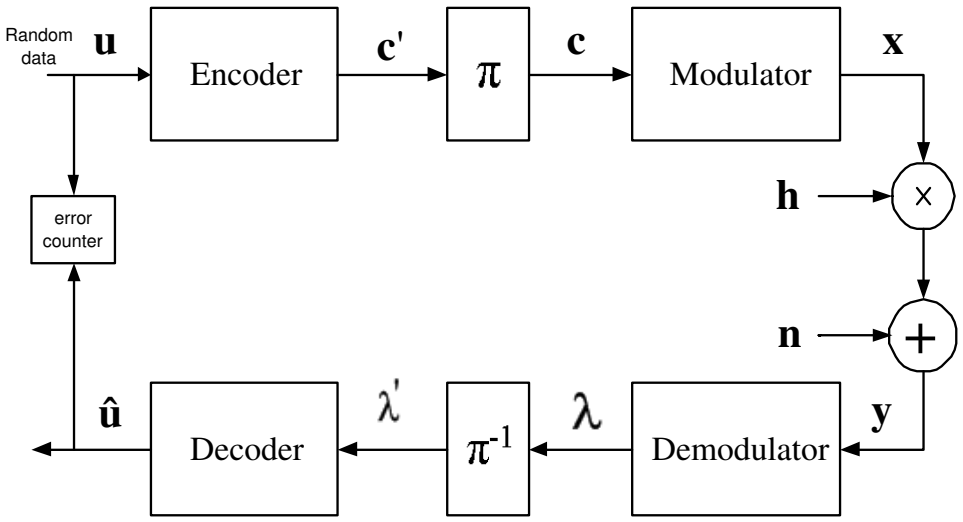

Figure 2.1: System model. π denotes interleaving at the bit [1] level.

able. In wireless mobile communications, a signal undergoes fading, which may be classified as large-scale fading or small-scale fading. Large-scale fading is due to propagation losses caused by obstructions over large areas. Small-scale fading is due to the small changes (as small as one half wave length) in the spatial positioning between a receiver and transmitter perhaps due to motion. When there are multiple reflective paths and there is no line-of-sight signal component, the amplitude of the small-scale fading coefficient $|h|$ is Rayleigh distributed, while the actual coefficient h is complex Gaussian with zero mean and independent real and imaginary components. On the other hand, when there is a significant line of sight signal component, then h is complex Gaussian with a non-zero mean and the magnitude of h is Rician distributed.

As shown in Fig. 2.1 the encoder is supplied with a binary message $\mathbf{u}$ and it generates a binary code word $\mathbf{c}'$. The code word is bit-wise interleaved to assure that bits in any code word fade independently [3]. With M-ary modulation, the bitwise interleaver assures that the bits mapped to each transmitted symbol are from different parts of the code word. The symbols are mapped to waveforms by the modulator and are transmitted over the channel. The transmitted signal $\mathbf{x}$ is affected by noise and fading when it propagates over the channel. At the receiving end, when the output of the demodulator is quantized to two levels, and fed to the decoder, it is called *hard-decision* decoding since the decoder depends on hard

decisions made by the demodulator. When the number of quantization levels used by the demodulator and fed to the decoder is greater than two, it is called *soft-decision* decoding [1].

When soft-decision decoding is used, each received symbol is transformed into symbol log-likelihood ratios for each possible signal $x_m, 1 \leq m \leq M$ in the signal set $\mathcal{S}$. Let $p(x_k|y)$ denote the conditional probability that $x_k \in \mathcal{S}$ was transmitted given that y was received. Let the likelihood function $f(y|x_k) = \kappa p(y|x_k)$, where κ is any multiplicative term that is constant for all x_k. Assuming that all symbols are equally likely, $f(x_k|y) \propto f(y|x_k)$ and the most likely symbol x_k is found by making a hard-decision on $f(y|x_k)$ or $\log\ f(y|x_k)$. Once the most likely symbol is identified, the corresponding bits can be determined by reversing the symbol-mapping function.

For example, consider the case of using QAM over an additive white Gaussian noise (AWGN) channel [1]. Let $y = x_k + n$, where n is complex i.i.d with mean zero and variance $N_o/2$ in each of the real and imaginary directions and the average energy per symbol is $E[|x_k|^2] = \mathcal{E}_s$. Then the log-likelihood for each x_k is found as

$$\begin{aligned} p(y|x_k) &= \frac{1}{2\pi\sigma^2} \exp\left\{\frac{-|y-x_k|^2}{2\sigma^2}\right\} \\ f(y|x_k) &= \exp\left\{\frac{-|y-x_k|^2}{2\sigma^2}\right\} \\ \log\ f(y|x_k) &= \frac{-|y-x_k|^2}{2\sigma^2} \\ &= \frac{-\mathcal{E}_s|y-x_k|^2}{N_o} \end{aligned} \tag{2.1}$$

Once the log-likelihood of symbol x_k is found for any signal set $\mathcal{S}$, then it can be trans-

[1] AWGN channels are characterized by h=1

formed into a log-probability using

$$\begin{aligned}
\Lambda_k &= \log\ p(x_k|y) \\
&= \log\ \frac{p(x_k|y)}{\sum_{x_m \in \mathcal{S}} p(x_k|y)} \\
&= \log\ \frac{f(y|x_k)}{\sum_{x_m \in \mathcal{S}} f(y|x_m)} \\
&= \log\ f(y|x_k) - \log \sum_{x_m \in \mathcal{S}} f(y|x_m) \\
&= \log\ f(y|x_k) - \log \sum_{x_m \in \mathcal{S}} \exp\left\{\log\ f(y|x_m)\right\} \\
&= \log\ f(y|x_k) - \max_{x_m \in \mathcal{S}} * \left[\log\ f(y|x_m)\right], \qquad (2.2)
\end{aligned}$$

where the pairwise max * function is defined as [4]

$$\begin{aligned}
\max*(x,y) &= \log\left[\exp(x) + \exp(y)\right] \\
&= \max(x,y) + \log\left(1 + \exp\left\{-|y-x|\right\}\right) \\
&= \max(x,y) + f_c(|y-x|), \qquad (2.3)
\end{aligned}$$

and f_c is a correction function that depends only on the magnitude of the difference between x and y. For multiple arguments, it follows the recursive relationship

$$\max*(x,y,z) = \max*\left[\max*(x,y), z\right]. \qquad (2.4)$$

2.2 Performance of the 802.11a Modulation

The IEEE 802.11a and 802.11g standards [2] use BPSK, QPSK, 16-QAM and 64-QAM modulation [2]. When these modulations are simulated, bits are chosen at random and grouped into frames. The frames are then passed through a bit-wise interleaver. The interleaved bits are then modulated using the constellation diagrams shown in Fig. 2.2. The simulation proceeds until a certain number of errors (50) are simulated or a low bit error rate (10^{-6}) is achieved. The simulated bit error rates of the modulation techniques used by the

[2] the only difference between 802.11a and 802.11g is that 802.11a operates in the 5GHz Unlicensed National Information Infrastructure (UNII) band, while 802.11g operates in the 2.4GHz band.

IEEE 802.11a standard in AWGN are shown in Fig. 2.3. As can be seen, BPSK and QPSK have identical bit error rate performance while 16-QAM is about 4dB worse and 64-QAM 8dB worse.

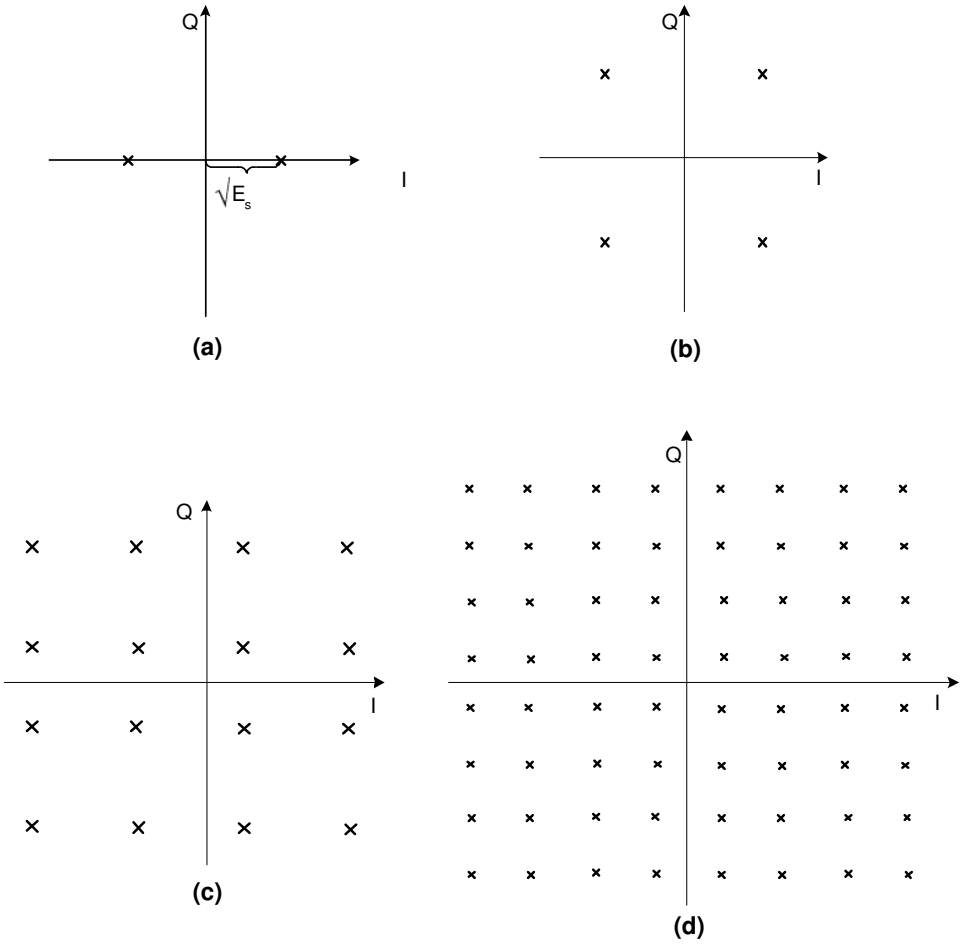

Figure 2.2: Signal constellations diagrams for:(a) BPSK (b) QPSK (c) 16-QAM (d) 64-QAM

2.3 Coded Modulation

The introduction of trellis-coded modulation by Ungerboeck [5] showed that combining modulation and coding is an efficient technique for improving performance. More generally, the combination of coding and modulation is called coded modulation (CM) [3]. Coding is performed over an expanded modulation signal set (relative to that needed for uncoded

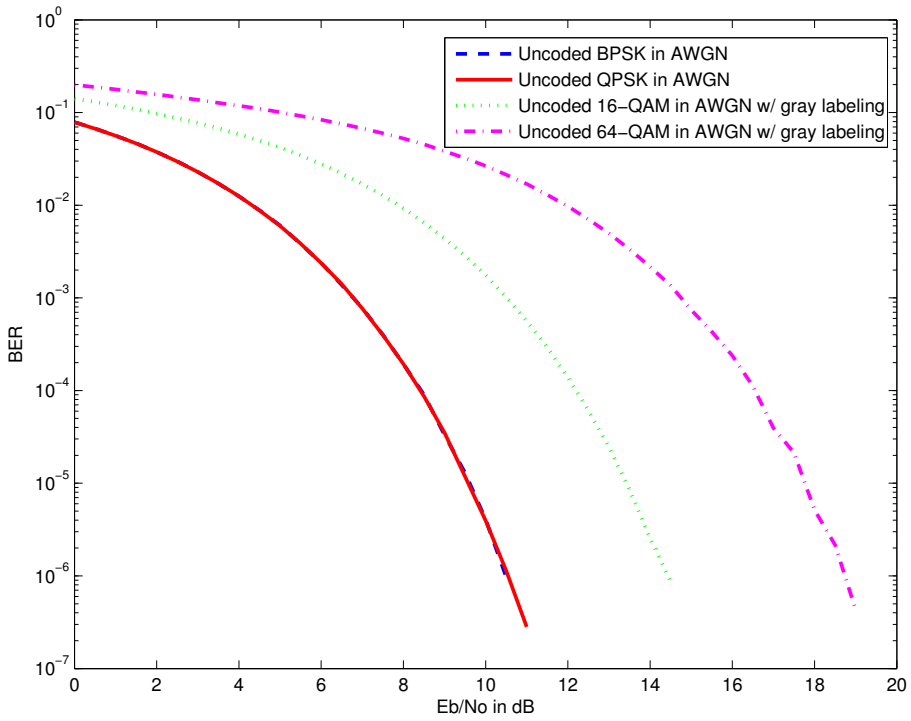

Figure 2.3: BER of uncoded BPSK, QPSK, 16-QAM, 64-QAM in AWGN

transmission) to achieve coding gain without bandwidth expansion or reduction in data rate. CM finds applications on bandlimited channels such as voiceband telephone, fixed terrestrial microwave links, satellite and mobile channels, but it is not generally used for mobile wireless applications.

2.4 Bit Interleaved Coded Modulation

Bit interleaved coded modulation (BICM) is the concatenation of a binary encoder with a memoryless modulator over a M-ary signal set with a bit-wise interleaver between the encoder and modulator [3]. It is sometimes referred to as "pragmatic coded modulation" [6]. BICM provides good performance in Rayleigh fading, especially when gray labelling is used [3]. The IEEE 802.11a standard uses BICM with gray labelling.

With BICM, the symbol log-likelihood must be transformed into bit log-likelihood ratios (LLR's) prior to being deinterleaved and decoded. Similar to the coded modulation receiver, the quantity $\log\ f(y|x_k)$ is calculated by the BICM receiver for each possible signal in $\mathcal{S}$. In

addition, the BICM receiver calculates the log-likelihood ratio of each code bit using:

$$\begin{aligned}
\lambda_n &= \log \frac{p(c_n = 1|y)}{p(c_n = 0|y)} \\
&= \log \frac{\sum_{x_k \in S_n^{(1)}} p(x_k|y)}{\sum_{x_k \in S_n^{(0)}} p(x_k|y)} \\
&= \log \frac{\sum_{x_k \in S_n^{(1)}} p(y|x_k)p[x_k]}{\sum_{x_k \in S_n^{(0)}} p(y|x_k)p[x_k]} \\
&= \max_{x_k \in S_n^{(1)}} * \log[f(y|x_k)] - \max_{x_k \in S_n^{(0)}} * \log[f(y|x_k)]
\end{aligned} \tag{2.5}$$

where $S_n^{(1)}$ represents the set of symbols whose n^{th} bit is a 1 and $S_n^{(0)}$ is the set of symbols whose n^{th} bit is a 0, and it is assumed that symbols are equally likely (i.e. $p[x_k] = 1/m$ for all $x_k \in \mathcal{S}$).

2.5 Channel Capacity

According to Shannon's channel coding theorem, every channel has associated with it a capacity C measured in bits per channel use [7]. The channel capacity is an upper bound on information rate r that can be transmitted with arbitrarily low error rate (for instance, 10^{-5}). The channel capacity of a memoryless channel is the mutual information between the channel's input X and output Y maximized over all possible input distributions [8]:

$$\begin{aligned}
C &= \max_{p(x)} \{I(X;Y)\} \\
&= \max_{p(x)} \left\{ \int \int p(x,y) \log \frac{p(x,y)}{p(x)p(y)} dxdy \right\},
\end{aligned} \tag{2.6}$$

The capacity of an AWGN channel is found by letting X be a zero mean Gaussian random variable with average energy $\mathcal{E}_s$ and letting $Y = X + N$, where N is zero mean Gaussian noise with variance $N_o/2$. The capacity of the channel is

$$C = \max_{p(x)} \{I(X;Y)\} = \frac{1}{2} \log_2 \left(\frac{2\mathcal{E}_s}{N_o} + 1 \right) = \frac{1}{2} \log_2 \left(\frac{2r\mathcal{E}_b}{N_o} + 1 \right) \tag{2.7}$$

where $\mathcal{E}_b$ is the energy per (information) bit.

Gaussian distributed input signals cannot be used in practice and therefore we are interested in determining capacity when the system is constrained to use signal set $\mathcal{S}$.

For a BPSK constrained input, $X = \pm\sqrt{\mathcal{E}_s}$, and the capacity is given by using $p(x) = (1-p)\,\delta(x+1) + p\,\delta(x-1)$, resulting in

$$\begin{aligned} C &= \max_{p(x)} I(X;Y) \\ &= I(X;Y)|_{p(x):p=1/2} \\ &= H(Y) - H(N) \\ &= \int_{\infty}^{\infty} p(y)\log_2 p(y)dy - \frac{1}{2}\log_2(\pi e N_o), \end{aligned} \tag{2.8}$$

where $p(y)$ is found by convolving $p(x)$ with $p(n)$. More generally, for any arbitrary modulation,

$$\begin{aligned} C &= I(X;Y) \\ &= E_{x_k,n}\left\{\log M + \log p(x_k|y)\right\} \ \text{ nats} \\ &= \log M + E_{x_k,n}\left[\Lambda_k\right] \ \text{ nats} \\ &= \log_2 M + \frac{E_{x_k,n}\left[\Lambda_k\right]}{\log(2)} \ \text{ bits} \\ &= \mu + \frac{E_{x_k,n}\left[\Lambda_k\right]}{\log(2)} \ \text{ bits}, \end{aligned} \tag{2.9}$$

where the expectation is taken over all symbols $x_k \in \mathcal{S}$ and all noise realizations N.

BICM [3] transforms the channel into μ parallel binary channels, and the capacity of the k^{th} binary channel is:

$$\begin{aligned} C_k &= E_{c_k,n}[\log(2) + \log\ p(c_k|y)]\text{nats} \\ &= \log(2) + E_{c_k,n}\left[\log\frac{p(c_k|y)}{p(c_k=0|y)+p(c_k=1|y)}\right]\text{nats} \\ &= \log(2) - E_{c_k,n}\left[\log\frac{p(c_k=0|y)+p(c_k=1|y)}{p(c_k|y)}\right]\text{nats} \\ &= \log(2) - E_{c_k,n}\left[\log\left\{\exp\ \log\frac{p(c_k=0|y)}{p(c_k|y)} + \exp\ \log\frac{p(c_k=1|y)}{p(c_k|y)}\right\}\right]\text{nats} \\ &= \log(2) - E_{c_k,n}\left[\max*\left\{\log\frac{p(c_k=0|y)}{p(c_k|y)}, \log\frac{p(c_k=1|y)}{p(c_k|y)}\right\}\right]\text{nats} \\ &= \log(2) - E_{c_k,n}[\max*\{0,(-1)^{c_k}\lambda_k\}]\text{nats}. \end{aligned} \tag{2.10}$$

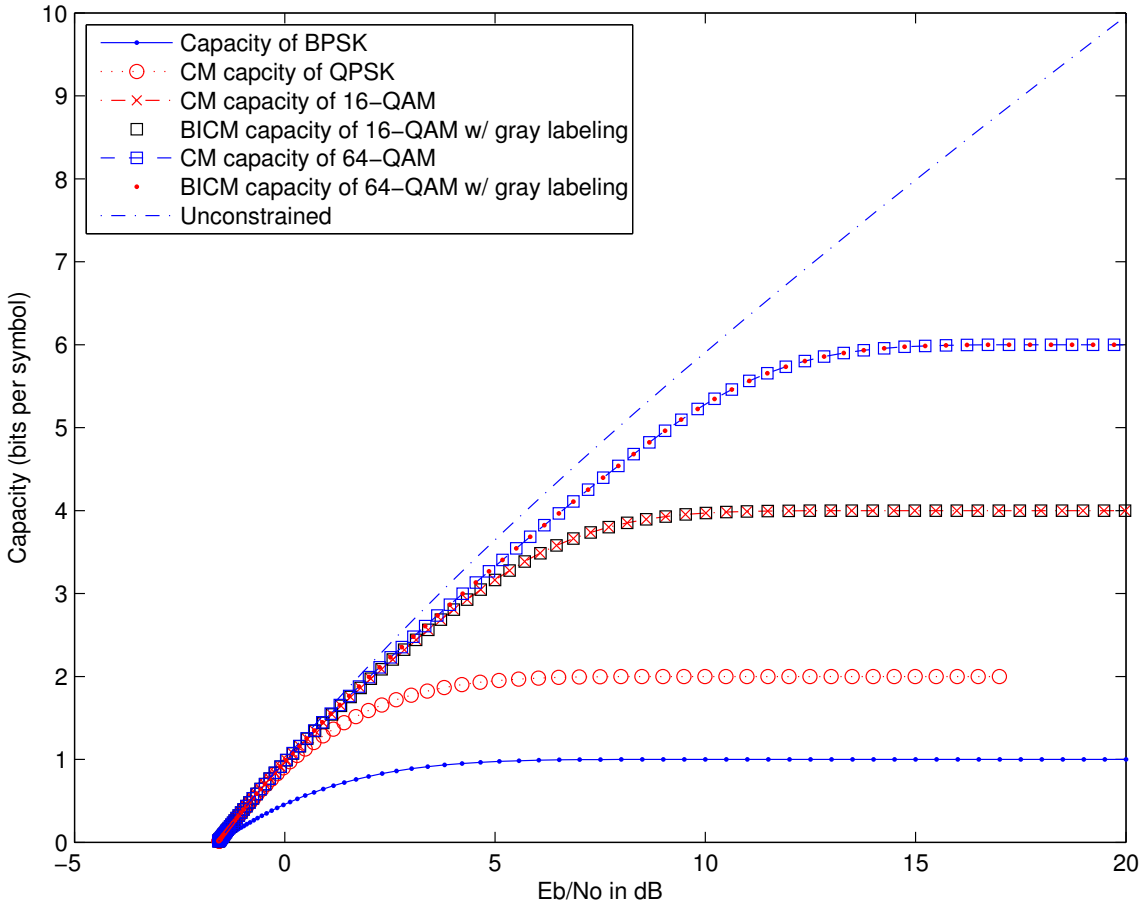

Figure 2.4: Capacity in AWGN with BPSK, QPSK, 16-QAM and 64-QAM

Since the capacity over parallel channels adds [8], the capacity of BICM is:

$$\begin{aligned} C &= \sum_{k=1}^{\mu} C_k \\ &= \sum_{k=1}^{\mu} \{\log(2) - E_{c_k,n}[\max*\{0,(-1)^{c_k}\lambda_k\}]\} \text{ nats} \\ &= \mu \log(2) - \sum_{k=1}^{\mu} E_{c_k,n}[\max*\{0,(-1)^{c_k}\lambda_k\}] \\ &= \mu - \frac{1}{\log(2)} \sum_{k=1}^{\mu} E_{c_k,n}[\max*\{0,(-1)^{c_k}\lambda_k\}] \text{ bits} \end{aligned} \tag{2.11}$$

The expectations in equations (2.9) and (2.11) can be obtained through Monte Carlo simulation. A signal x_k is drawn randomly from the signal set $\mathcal{S}$ and then added to random noise. The received signal is passed through the demodulator. For CM, the symbol log-

probability Λ_k is used with equation (2.9) to obtain the capacity, while with BICM the bit log-likelihood ratio λ_k is used with equation (2.11). This type of simulation permits us to study higher dimension signals which are otherwise not possible using purely analytical approaches. The Monte Carlo simulation of the capacity in AWGN for BPSK, QPSK, 16-QAM and 64-QAM is shown in Fig. 2.4. It can be observed in the case of 16-QAM and 64-QAM, the capacities of CM and BICM are almost the same when gray labelling is used.

2.6 Convolutional Codes

Convolutional codes are used in real-time applications because they can be continuously encoded and decoded using a soft-decision decoder. Convolutional codes are commonly specified using the notation (r, K) where the quantity $r = k/n$ is the *code rate*, (n is the number of output bits and k is the number of input bits). The *constraint length* K is the number of input bits that affect each output bit and the encoder has $K - 1$ delay elements. Codes of rate $1/n, n \geq 2$, can be used to produce codes with rates higher than $1/n$ by selectively deleting bits through a process called puncturing. Such codes are called *punctured* codes and families of convolutional codes that are punctured from the same mother code are called *rate compatible punctured convolutional* codes [9].

Convolutional codes can be represented as state diagrams (for analyzing performance) and trellis diagrams (for visualizing the decoding process). A trellis diagram for a 4 state convolutional code is shown in Fig 2.5. Decoding of convolutional codes usually follows the maximum likelihood decoding rule which involves choosing the code sequence through the trellis that has the smallest Hamming distance to the received sequence for hard-decision decoding and smallest Euclidian distance for soft-decision decoding. The Viterbi algorithm [10] is a computationally efficient way to perform maximum likelihood decoding by finding the best state sequence one stage at a time in the trellis.

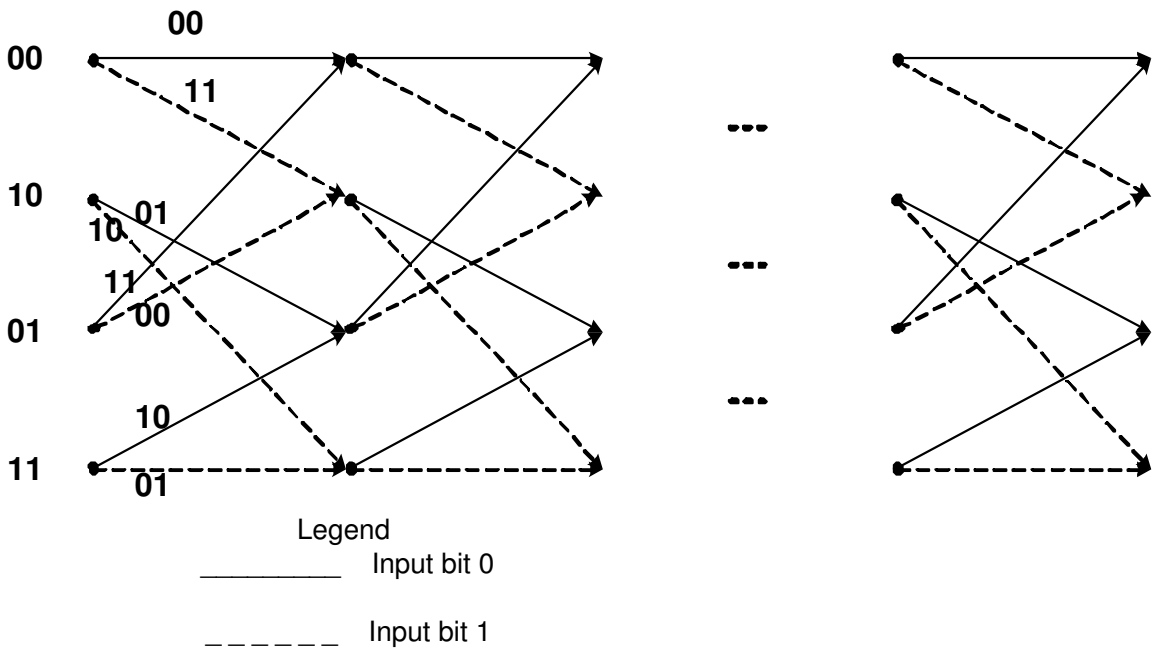

Figure 2.5: A trellis for a 4 state convolutional code

2.7 Coded Performance of 802.11a

The IEEE 802.11a standard uses a convolutional code with constraint length $K = 7$ and base rate $r = 1/2$, though higher rates are achieved through puncturing [2]. The modulation techniques and code rates employed for different data rates of the IEEE 802.11a protocol are listed in table 2.1. The generator polynomials used by the IEEE 802.11a standard are $g_o = 133$ and $g_1 = 171$ in octal. When the generator polynomials are expanded into binary vectors and placed into a matrix, the result is,

$$g = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{2.12}$$

The encoder is as shown in Fig. 2.6. The puncturing patterns used to generate code words with rates 2/3 and 3/4 are

$$p_1 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \tag{2.13}$$

Table 2.1: Rate dependent parameters for IEEE 802.11a.

Data rate(Mbps)	coding rate	Modulation
6	1/2	BPSK
9	3/4	BPSK
12	1/2	QPSK
18	3/4	QPSK
24	1/2	16-QAM
36	3/4	16-QAM
48	2/3	64-QAM
54	3/4	64-QAM

$$p_2 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}. \tag{2.14}$$

In the above puncturing patterns, the presence of a 1 indicates that the bit is transmitted, while the presence of a 0 indicates that the bit is deleted.

In the simulations, the frame size used is 5114. The SNR was varied in increments of no more than 1dB, and for each SNR value, enough trials were run to generate 530 independent frame errors. The simulated BER for all eight rates specified in the IEEE 802.11a standard in AWGN and Rayleigh fading environment are shown in Fig 2.7 and Fig 2.8 respectively. It can be observed that BPSK and QPSK have the same performance, and therefore the performance of 6 and 12 Mbps, 9 and 18 Mbps is almost the same. There is a trade-off involved between energy efficiency and bandwidth efficiency. To achieve higher data rates, more transmit power is required.

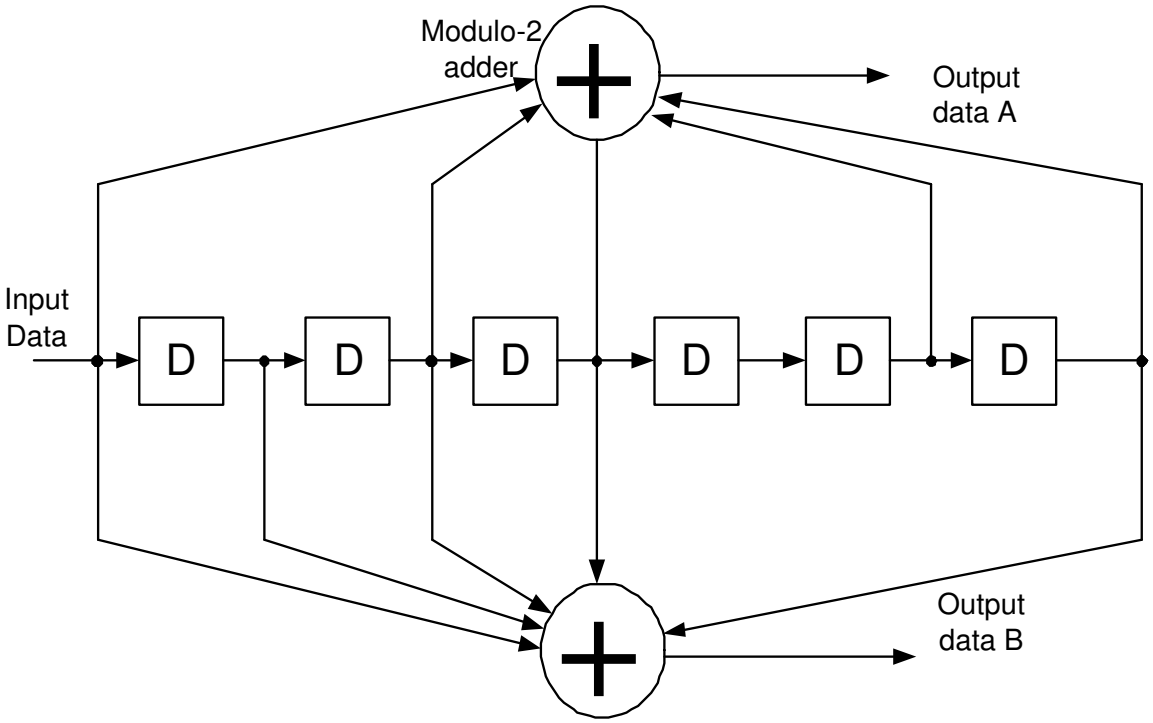

Figure 2.6: Convolutional encoder with K=7 and r=1/2 used by the IEEE 802.11a standard.

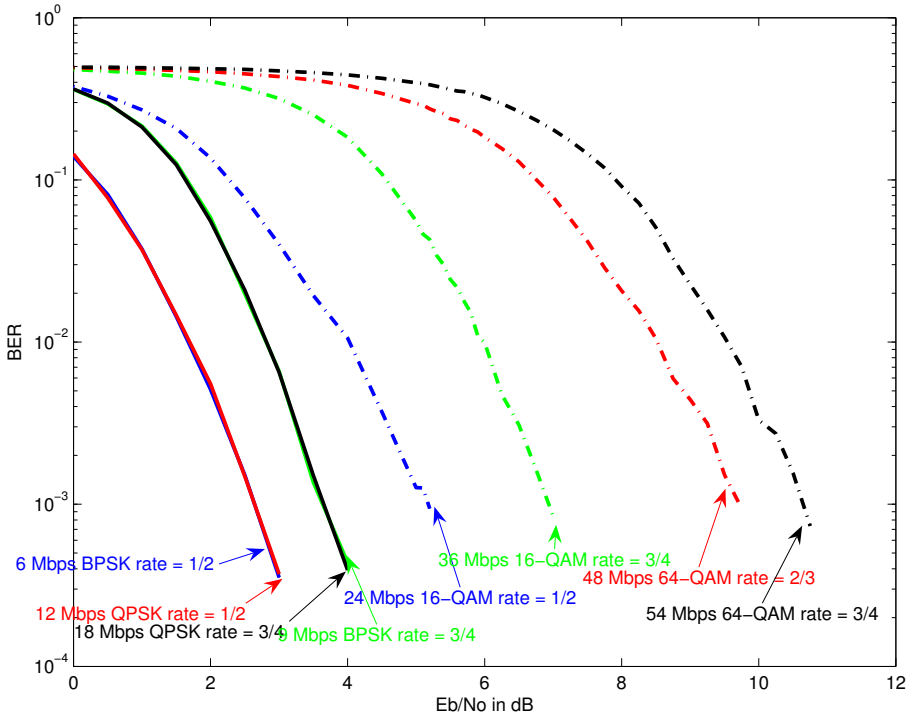

Figure 2.7: BER of IEEE 802.11a with convolutional coding in AWGN

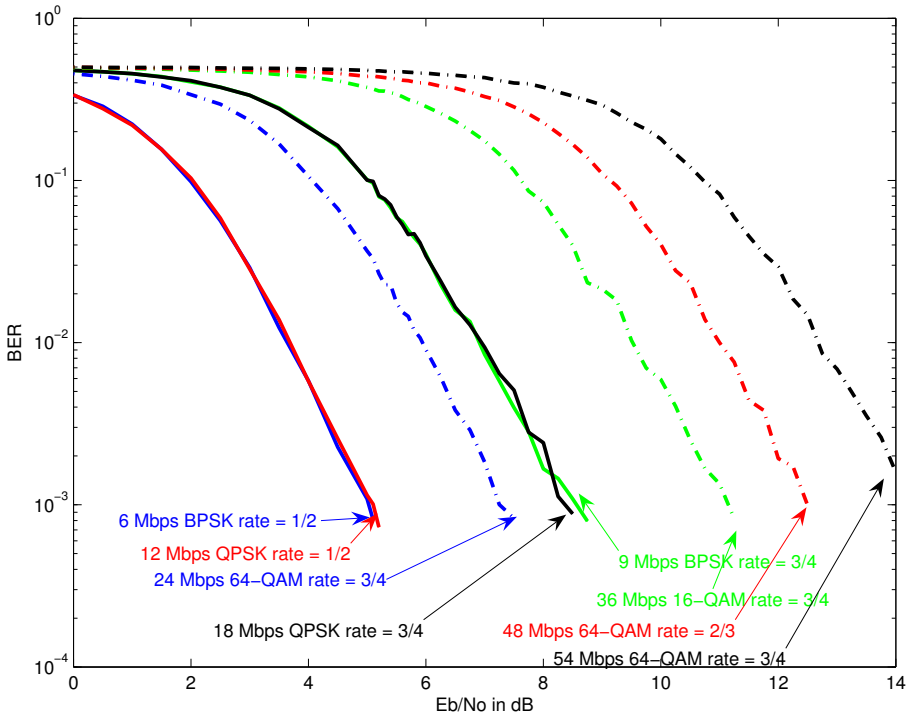

Figure 2.8: BER of IEEE 802.11a with convolutional coding in Rayleigh fading

Chapter 3

Grid Computing: An Overview

This chapter provides a brief history of distributed computing and introduces the various supercomputing technologies from which grid computing has evolved. This is followed by a discussion of the different grid models including desktop grids.

3.1 Brief History of Grid Computing

Grid computing technologies enable large-scale aggregation and sharing of computational, data and other resources across organizations [11]. Before grid computing was introduced, various distributed computing research projects were carried out by communities to solve compute intense problems. In the early 1970's when networks were first formed (before the formation of the Internet), a project was conducted on the Advanced Research Projects Agency Network (ARPANET), the predecessor of the Internet. It tested the idea of harnessing unused CPU cycles using two distributed computing programs called *creeper* and *reaper* [12].

John F. Shoch and Jon A. Hupp of Xerox's Palo Alto Research Center (PARC) conducted another distributed computing experiment in 1973. In the experiment, a worm used the idle CPU cycles of the machines that were connected in a local Ethernet network for rendering computer graphics. The first Internet based distributed computing project was started by Digital Equipment Corporation (DEC) in 1988. In this project, tasks were sent through email to volunteers, who would run the tasks during idle times and send back the results.

The grid computing projects underway today form a grid on a large scale that facilitates collaboration on scientific projects utilizing geographically distributed resources. In 1995, at the IEEE/ACM Supercomputing conference in San Diego, high speed networks were used to connect, for a short time, geographically distributed high-performance computers and advanced visualization environments. This experiment, called the I-way (Information Wide Area Year), was led by Ian Foster of the United States Department of Energy's Argonne National Labs and University of Chicago [13]. This experiment laid the foundation for the Globus project to develop the Globus tool-kit a *de-facto* standard accepted by many research and commercial communities to implement Grid computing [14].

3.2 Supercomputing Technologies

The various supercomputing technologies are:

- **High-Performance and Cluster Computing**

  High-performance computing is a class of applications used in advanced science and engineering disciplines to provide very high processing power by efficiently utilizing supercomputers. High-performance computing resources provide over an order of magnitude more computing power than is normally available on ones desktop. These are used to solve grand-challenge problems in areas like computational fluid dynamics (CFD) and finite-element methods (FEM) [15].

  Clusters are built using commodity-off-the-shelf (COTS) hardware components and run open-source software like the Linux operating system and are used to solve complex problems. The processors are often connected using high speed interconnects like Infiniband or high speed gigabit Ethernet [16].

- **Peer-to-Peer Computing**

  Peer-to-peer computing is a paradigm that enables sharing of storage devices across the Internet [17]. In this model, users can share PC folders with groups of other users over the network [18]. The peer-to-peer model is used for immediate and temporary information sharing between users and demands more network and administration

resources because each computer can act as a server and so the network resources are consumed quickly.

- **Internet Computing**

  Internet computing harnesses idle workstations and PC's by creating powerful distributed computing applications [11] to provide Supercomputing resources.

  Some popular Internet computing projects and Internet computing companies are as follows :

  Internet Computing projects:
  - SETI@home setiathome.ssl.berkeley.edu
  - Folding@home www.stanford.edu/group/pandegroup/Cosm
  - Compute-against-Cancer www.parabon.com/cac.jsp
  - Fight AIDS@home www.fightaidsathome.org
  - Great Internet Mersenne Prime Search mersenne.org
  - Casino 21: Climate simulation www.climate-dynamics.rl.ac.uk

  Internet computing companies:
  - Entropia www.entropia.com
  - United Devices www.uniteddevices.com
  - Parabon www.parabon.com
  - Popular Power www.popularpower.com

3.3 Grid Computing

Grid computing is a technology that has evolved from the supercomputing technologies mentioned in Section 3.2 to enable flexible, controlled resource sharing on a large scale. The definition of grid computing given in [19] is as follows: " grid computing enables virtual organizations [20] to share geographically distributed resources as they pursue common goals, assuming the absence of central location, central control, omniscience and an existing trust relationship". Some popular grid projects are:

- Grid Physics Network project www.griphyn.org
- European Data Grid grid.web.cern.ch/grid
- Particle Physics Data Grid www.ppdg.net
- Network for Earthquake Engineering Simulation Grid www.neesgrid.org
- The Globus Project www.globus.org
- World Community Grid www.worldcommunitygrid.org
- The Global Grid Forum www.gridforum.org

The applications that require high computing power can be classified based on the granularity of the processes performed as shown in Fig. 3.1 [19]. In this context, granularity refers to the size of the computation that can be performed between communication points. Some applications in science and engineering can be implemented to be solved in a sequence (Fig. 3.1, upper left). These serial applications can be solved using isolated desktop machines, servers and supercomputers.

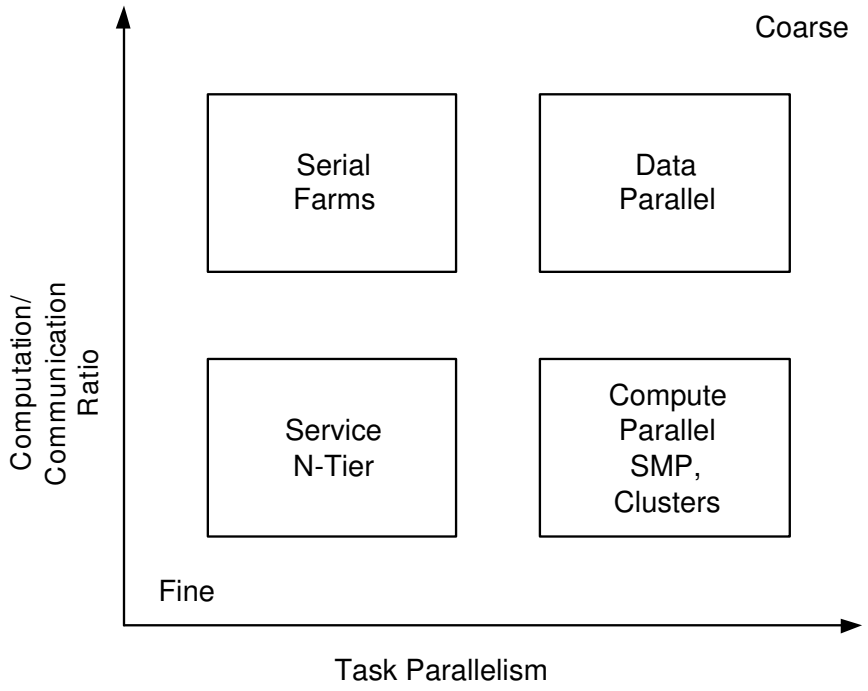

Figure 3.1: Application and architectures

The type of applications that involve performing the same operation on a large data set that can be executed independently in parts, exploiting the parallelism in the data are termed *embarrassingly parallel* applications. These type of applications are executed by first

dividing the input data into parts and independent processing of the data segment using the same executable. Individual results are grouped to produce the output.

These data parallel applications (Fig. 3.1, upper right) that are normally executed on compute *farms* can be efficiently executed using desktop grids explained in Section 3.3.1 by harnessing idle CPU cycles. Various applications in the industry that have successfully implemented the embarrassingly parallel applications in this kind of environment are genome sequencing in the life sciences, Monte Carlo simulations in High Energy Physics, risk analysis in financial services, etc.

The other type of applications are those that do not have data parallelism but have parallelism at the code level called compute parallel applications (Fig. 3.1, lower right). These applications are executed on cluster computers with either shared or distributed memory. In parallel programming, threads are used in shared memory context as in OpenMP and Message Passing Interface (MPI) is used in distributed memory systems. Distributed memory parallel computing applications (Fig. 3.1, lower left) can communicate extensively using MPI and are generally used to solve Grand challenge problems on high performance computers.

3.3.1 Desktop Grids

A typical PC has performance exceeding that of a multi-million dollar supercomputer a few decades ago. Cycle stealing technologies that harness the idle CPU cycles of desktop PC's and workstations are widely used in various projects as listed in Section 3.3 such as SETI@home which harvests the compute power of over a million CPU's [11].

There are millions of desktop systems around the world that are used for only about 2 or 5 percent of their capacity. The idle capacities are a great resource, if they could be tapped. A *desktop grid* is a computing infrastructure that utilizes unused processing power of desktop resources. It enables the availability of the processing power to an end user in a transparent way by hiding the complexity of managing the jobs.

Enterprise applications are developed by various companies like Entropia, United Devices and Parabon that promise computational power to an organization without any huge additional infrastructure costs by being able to harness the idle cycles on the machines already

available within their organizations.

Some desired attributes of a desktop grid are:

1. Security: The integrity of the distributed computation should be protected [21]. The resource provider should not have any access to the jobs that use the idle time of the resources. Also, the jobs that are performed on the grid should not have access to resources other than those specified and agreed initially.

2. Authority: The desktop resources should be accessible to be utilized in the grid only when they are idle or a portion of the processing power could be used always with the level of priority specified.

3. Centralized control: There should be central control by a server which manages the allocation of resources, submission of jobs and collection of results. It may also incorporate redundancy features for accuracy and to overcome the undependable nature of the resources.

4. Scalability: The desktop grid should have the provision for inclusion of new resources on a large scale.

Desktop grids are best suited for embarrassingly parallel problems that have very little or no inter process communication (high compute to communication ratio). The Global Grid Exchange is a desktop grid that is used in this work and is detailed in the next chapter.

3.3.2 Cluster Grids

Clusters which are autonomous and are geographically distributed can be grouped to form a *cluster grid*. They allow the sharing of resources and virtual organizations can be formed to pursue a common goal. Processors in a cluster communicate using Message Passing Interface (MPI) and MPI-G (grid) can be used for communication between clusters in a cluster grid.

Clusters use software to provide resources to the end user in a transparent way by using

either a single system image or a single system environment. A single system image requires the modification of the Linux operating system kernel. The software that virtualizes the group of clusters into a single entity provides the process identifiers for running the jobs over the cluster grid resulting in a distributed process space. A single system environment does not require any kernel modification. This approach uses front end software that enables the end user to launch multiple processes over the cluster grid.

The Saber grid is an example of a cluster grid formed by the Pittsburgh Supercomputing center and West Virginia University to provide Supercomputing resources to researchers in Morgantown and Pennsylvania. To access the Saber grid, users need to securely connect (ssh) to intel2.psc.edu or energy.cluster.wvu.edu.

3.3.3 Data Grids

Data grids enable users to securely access remote data resources such as flat-file data, relational data and streaming data. A data grid provides transparent, secure, high-performance access to data across administrative domains and organizations. The administrative unit of a data grid is the *grid domain*. A data grid consists of one or more grid domains. The ability to create multiple grid domains and interconnect them enables a decentralized administration of the grid.

The major components of a data grid are *grid servers* that perform domain creation, authentication, access control and monitoring of resources. The first grid server deployed starts the data grid and this grid server is also called the *Grid Domain Controller* (GDC). Starting multiple grid servers enables the construction of large data grids where each grid server can handle a part of the grid.

An example of a data grid is the *TeraGrid* [21]. It accumulates resources at eight partner sites to create an integrated, persistent computational resource. It gives 40 teraflops of computing power and nearly 2 petabytes of rotating storage. It is coordinated through the Grid infrastructure group at University of Chicago Argonne National Laboratory working in partnership with eight other organizations. One virtual organization that makes use of the Tera Grid is the National Virtual Observatory (NVO) [21]. It uses the Grid to expose

large data to massive computing, run representative application that explore data, foster new projects in astronomy that use NVO's services and TeraGrid resources.

Chapter 4

Simulation of IEEE 802.11a using Global Grid Exchange

An initiative of the West Virginia High Technology Consortium Foundation (WVHTCF), the Global Grid Exchange is a secure grid computing platform powered by the Frontier software developed by Parabon Inc. It provides access to massive computational resources to quench the ever increasing demand for computational power in science and engineering applications by aggregating the idle or unused CPU cycles in the available resources throughout the state of West Virginia. The key aspects of the Frontier are its ability to provide access to bursts of processing power and super computing resources at an affordable cost in a secure manner.

4.1 The Frontier Platform

4.1.1 Introduction

The definition of Frontier is given in [22] is as follows: "Frontier is a massively scalable, distributed computing platform built on the foundation of Internet computing. Specifically, Frontier draws on the otherwise unused computational capacity of relatively low-power, unreliable, high-latency nodes in the form of Internet-connected desktop computers running in parallel to create a coherent, high-power, reliable whole". [22].

Frontier provides the framework to define, launch, execute and manage parallel applica-

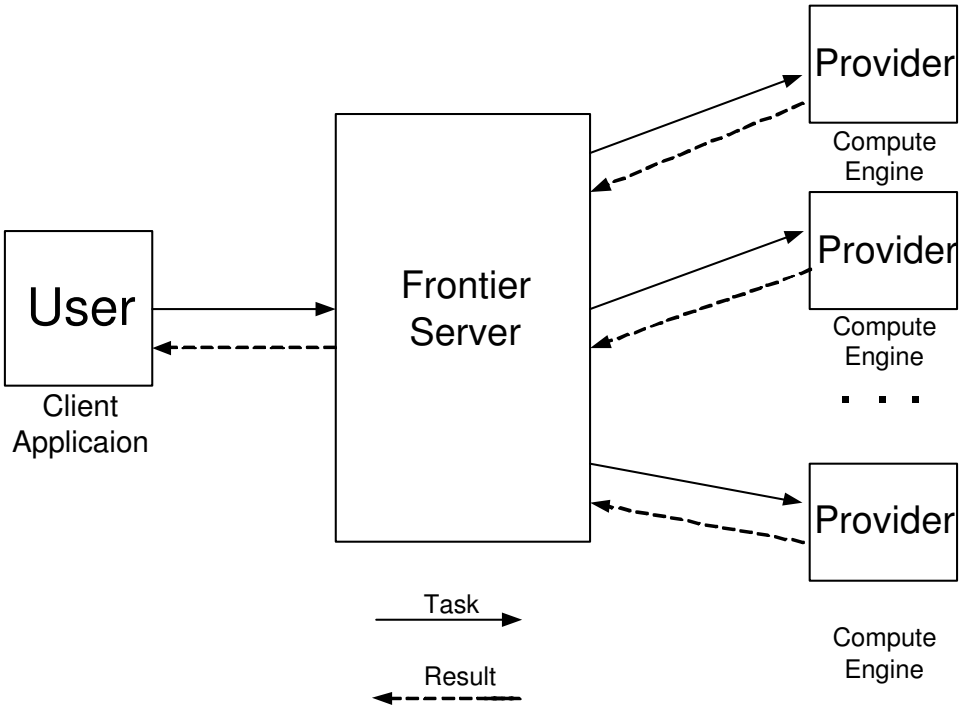

Figure 4.1: Frontier Platform

tions across the Internet. Frontier harnesses the unused power from low-power, unreliable and high latency nodes which are mainly desktop PC's on a large scale by using their idle time. Generally, Frontier runs the jobs under Java Virtual Machine (JVM) requiring that the programming language used is Java or any language that produces bytecode. It also has a "native code" option for running compiled executables and this aspect is detailed in section 4.3.

The Frontier platform is made of the following: *The Client Application*, an application that runs on a user's system and is used to connect to the Frontier server. (Jobs can be launched, monitored and terminated using this application); *The Frontier Compute Engine*, an application that runs on a machine utilizing its idle cycles to provide computational power to the grid users; and *The Frontier Server*, which handles the assignment, allocation of resources, scheduling and management of jobs on the grid.

4.1.2 Frontier Terminology

The terminology used by Frontier is as follows:

- **Jobs and tasks**

  A *job* is a set of tasks and is defined by a set of elements it contains, a set of elements it requires, a list of parameters and an entry point in the form of a Java class name. The tasks have no inter-task communication capability. All the communication to a task takes place from the server at the time when the task is instantiated and all the communication from a task takes place via status reports and implicit check points. A *parameter* list is a set of name-value pairs. A *name* is a text string and the *value* is a pre-defined data type. Parameter lists can be conveniently grouped together into parameter maps to be attached to task specification.

- **Data and executable elements**

  Data elements are blocks of data represented as a parameter list, sent to a compute engine for execution of a task. *Executable elements* provide the Java byte code in a jar file that defines the computation to be performed by the Frontier compute engine.

- **Task status reports**

  The results from a task execution are reported to the central Frontier server and finally to the client application in the form of status reports from the Frontier compute engine. The different types of status reports are *run mode* (unstarted, running, complete,... indicating the status of the task), *results* (results are returned as name-value pairs), *exceptions* (code and text descriptions of exceptions returned while a task is executed), *progress* (a single scalar metric indicating the progress of a task).

- **Checkpoints**

  A task running on a machine may not complete in a single session due to the unreliable nature of the resources. The resources utilized in the grid are autonomous, the running task will be aborted if the user moves his mouse or uses his desktop resulting in the loss of all the unreported results obtained until that time. This vulnerability may prevent

the task from running to completion. Frontier uses *checkpoints* to save the state of a task at pre-defined intervals by the Frontier compute engine and sending the state variables to the Frontier server. The server can reschedule the task for execution from the last checkpoint.

4.1.3 Design Issues

The considerations that affect the design and implementation of the Frontier platform in developing applications are:

1. **Basic computation should be valid Java bytecode**. The Java programming language is used for developing applications owing to it's inherent security features provided by the Java Virtual machine(JVM). Similar to the *sandbox* technology provided by Java that provides security to run Java applets the Frontier compute engine ensures that the tasks are securely executed in a restricted environment without any access to the system resources such as disk access and network access, other than those agreed upon initially. The part of the client application that communicates with the Frontier server must be written in Java and the rest of the application can be completely developed in any programming language that can eventually be compiled to Java bytecode.

2. **Optimal size of the tasks that constitute a Job**. The task size should be optimized such that the computation to be performed by the task is effective when using the available system resources and the time required by an independent task is short(a few hours or few days). There should be a high compute to communication ratio.

3. **Launch and listen**. Jobs can be launched using the client application to connect to the Frontier server. Due to the unreliable nature of the grid resources, the time required for the completion of a job is very unpredictable requiring that the application should continue to run for a very long time. Frontier obviates the need for the client application to continue running until it receives the results by being able to connect

to the server and retrieve the results at a later time after the client application has exited.

4.1.4 The Frontier Application Program Interface (API)

The Frontier API is used for initiating task execution, setting task parameters, accessing data elements, reporting results and logging checkpoints. This section intends only to introduces the various components of the Frontier API. For complete detailed information, please refer to the Frontier API documentation [22].

Tasks, the basic units of execution are submitted to Frontier server using the client API which is discussed in the later part of this section. The client application is used to monitor the task and also obtain the results.

Intermediate results obtained by the Frontier compute engine during the execution of a task can be reported to the Frontier server and the frequency at which this is performed should be set by considering a trade-off between the result information made available to the user versus slowing down the task execution. Checkpoints can be logged by modifying the task specification by replacing the previous parameter sets with newer ones. The frequency at which this is done is a trade-off between lost computation time, disk space and the risk of losing results when a task is terminated. The progress of each task is saved as a positive scalar by the compute engine and can be reported more frequently than results to the Frontier server without any significant communication overhead. A sample diagram of the job controller that is used to monitor the jobs launched over the grid is shown in Fig. 4.2

The client API is used for creating, launching, monitoring and controlling jobs and tasks on the grid. The client application can be used either in remote or in local mode. In remote mode the client application connects to the Frontier server and submits the jobs to be run on the grid. In local mode, the client application does not connect to the Frontier server and creates a virtual session, executing the jobs on the local machine.

The jobs and tasks that are created and submitted to the Frontier server using the client application are identified by assigning attributes like the registered name of the machine that

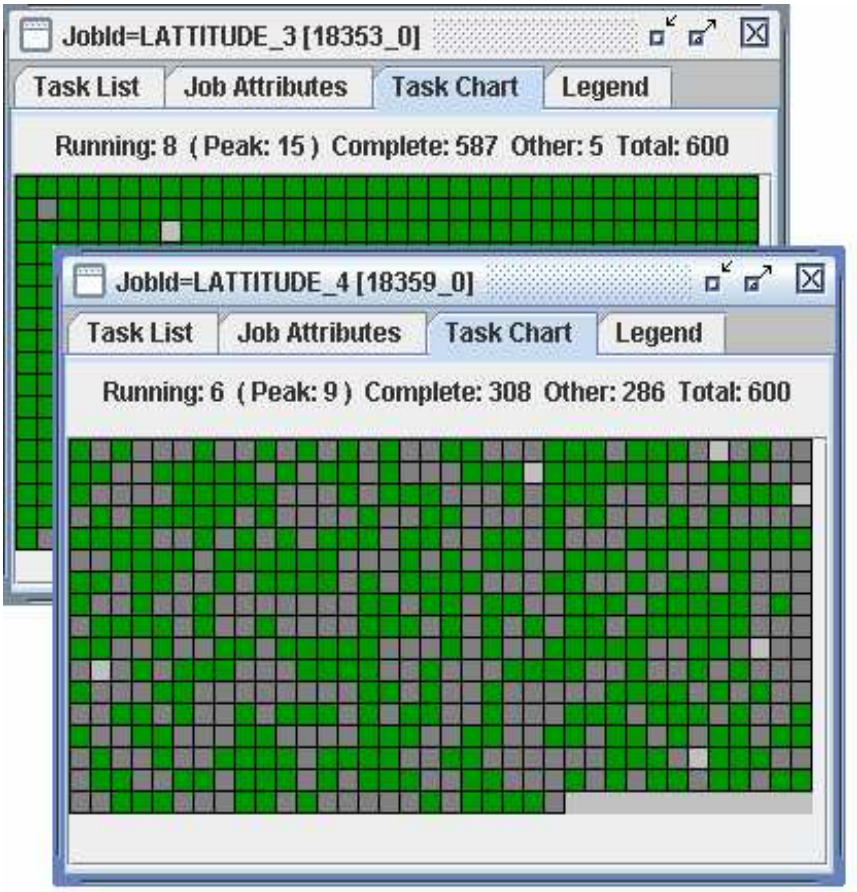

Figure 4.2: Job controller

launched the job, the launch date of a task or a set of email addresses to which the results are to be sent when the job is complete. In our work, the registered names of the machines along with an incremental job number and task number was used to identify the jobs and tasks.

A listener should be running for a job, to obtain the results of the tasks within a job, to know the status of a job and also to remove a job when it is complete. In cases, when a listener is not running for a job and information related to a job exists on the server and is not available on the local machine, it is referred to as released. The listener needs to be started again to connect to the server to obtain the information, called re-establishing.

4.1.5 Security

To ensure the security of the provider, the Frontier platform primarily supports only the Java programming language which has inherent security features. It also supports native code as detailed in Appendix-A. The tasks executed over the grid cannot access the data and programs present on the provider's machine. The inherent vulnerability by a virus to a provider's machine connected to the Internet is not increased with the installation of Frontier. Frontier uses the secure sockets layer (SSL) protocol developed by Netscape and accepted as the Transport Layer Security (TLS) by the Internet Engineering Task Force (IETF) as a secure and authentic means of communications over the Internet.

Also, all the information pertaining to the grid user other than the computation to be performed is removed from the jobs before they are sent to the provider's nodes. The tasks that are sent over the grid are essentially shredded and so a provider's node receives little pieces of various problems, making it almost impossible to do reverse engineering to find the complete job details. The Global Grid Exchange runs tasks with known results to check for any malicious activity at the provider's node. If the result returned from the provider's node is not correct, that provider is no longer used in the grid.

4.2 Stand-alone Applications using Matlab

Stand-alone applications are created for the programs written in Matlab using the Matlab compiler. This obviates the need for a Matlab license on the provider machines that are being harnessed in the grid and it only necessitates that Matlab component runtime[1] is installed to execute them. Also, the performance of the executables is better compared to that of the Matlab programs, since the Matlab compiler compiles the Matlab programs into executable binary low level codes, that run without the need of the Matlab interpreter. The stand-alone application consists of an executable file and another component technology file (CTF) archive that contains the files needed by the component for execution at run-time. For more information about developing stand-alone applications using Matlab, please refer to the Matlab documentation.

[1] A stand-alone set of shared libraries that enables the execution of M-files

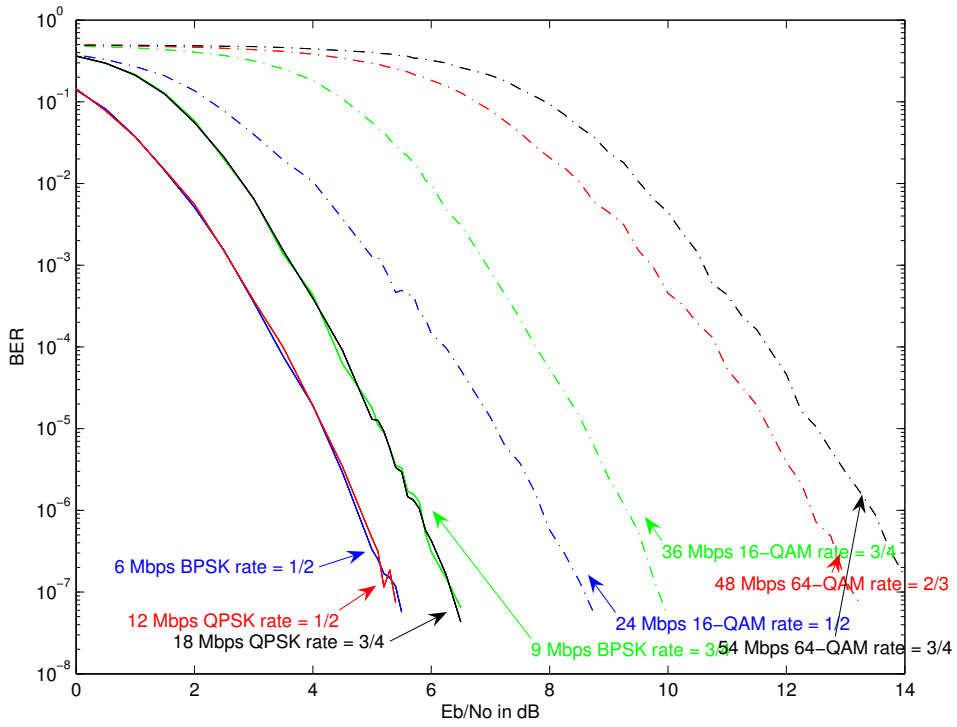

Figure 4.3: BER of IEEE 802.11a in AWGN, simulated on the grid

4.3 Implementation

The compiled Matlab code runs as native code. The Frontier compute engine needs to be enabled to run applications containing native libraries. The ability to run native code is controlled by the provider and also by the Frontier administration. The provider has to enable the settings of the compute engine to run native code. A grid user by default is not permitted to run native code on the Frontier platform and has to be granted a privilege by the grid administrator. The tasks written in Java interact with the native libraries using Java Native Interface (JNI). The grid users running native code on the Frontier platform can be restricted to run their applications on a specific node group provided by the user. For more information about issues related to running native code on Frontier, please refer to the user notes "Running native code on Frontier" of the Frontier technical documentation provided in the Appendix A.

The two files, one file containing setup information (not needed to be reported back), another file containing the state information (needed to be reported back) along with the

stand-alone application are sent to the Frontier compute engine using the client application via the Frontier server. The frontier compute engine sends a data file containing the results to the client application via the Frontier server.

Checkpointing mechanism has been implemented using two strategies. At the executable level, the state of the simulation is periodically saved to a file for the simulation to be resumed from that point when it re-starts. At the job level the compute engine can save the task specification with the frequency level specified to log a checkpoint, when interrupted. For more information about the implementation refer to Appendix B.

4.4 Simulation Throughput of the Grid

Simulation throughput is the number of simulations executed on a system per unit time. The rate of execution of tasks on grid were monitored using an utility called CmlGridMonitor. It periodically notes the number of trials simulated on grid by observing a saved file to

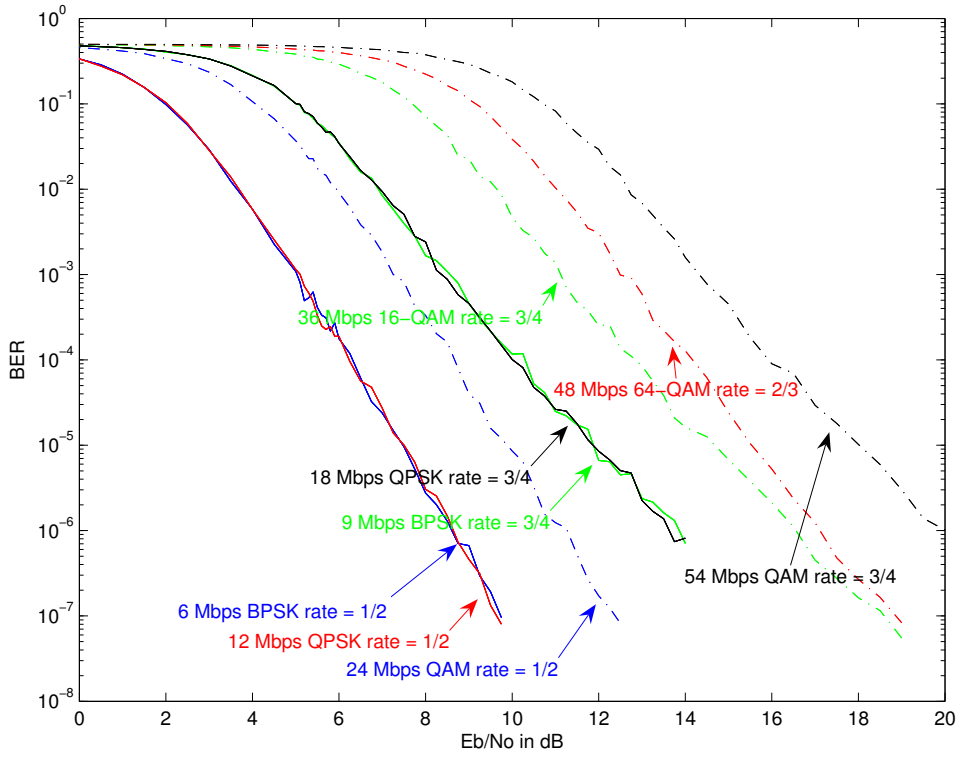

Figure 4.4: BER of IEEE 802.11a in Rayleigh fading, simulated on the grid

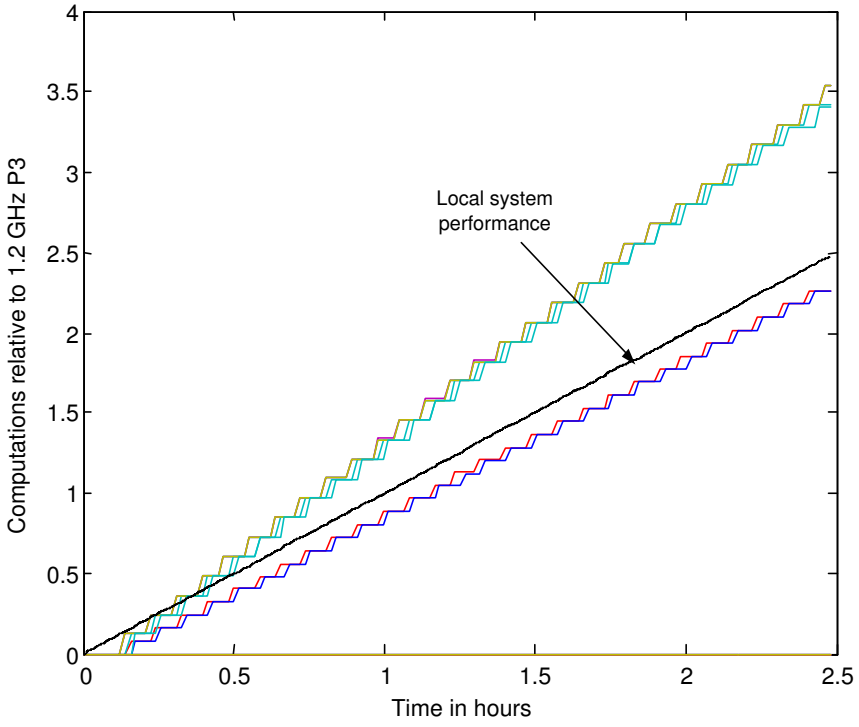

Figure 4.5: Performance plot-1, 7 tasks ran in parallel on the grid, 2 were slower than the local machine and 5 were faster

determine the total number of trials performed on the grid. The throughput is normalized with reference to a system with a 1.2GHz PIII, w/ 512 Mbytes RAM and running windows 2000 version.

The progress of a simulation is shown in the Fig. 4.5 in which 7 tasks ran in parallel on the grid, 2 were slower than the local machine and 5 were faster. After 150 minutes, the local computer executed 159,013 trials, while the grid executed 1,408,483, nearly an order of magnitude improvement. Dotted black line shows performance of local laptop, a 1.2 GHz PIII w/ 512 Mbytes RAM, which processes 64,140 simulation trials per hour. Fig. 4.6 shows the progress of another simulation in which 11 tasks ran in parallel, 1 was faster than the local machine (gold line), 9 were slightly slower and 1 was significantly slower (red line). After 9.5 hours, the grid executed 6,019,410, nearly an order of magnitude improvement over running locally. Similarly, Fig. 4.7 shows the complete simulation of the sample shown in Fig. 4.6. It can be observed that 11 tasks ran in parallel and after 24 hours, the grid executed 16,630,510 trials, an order of magnitude improvement over running locally.

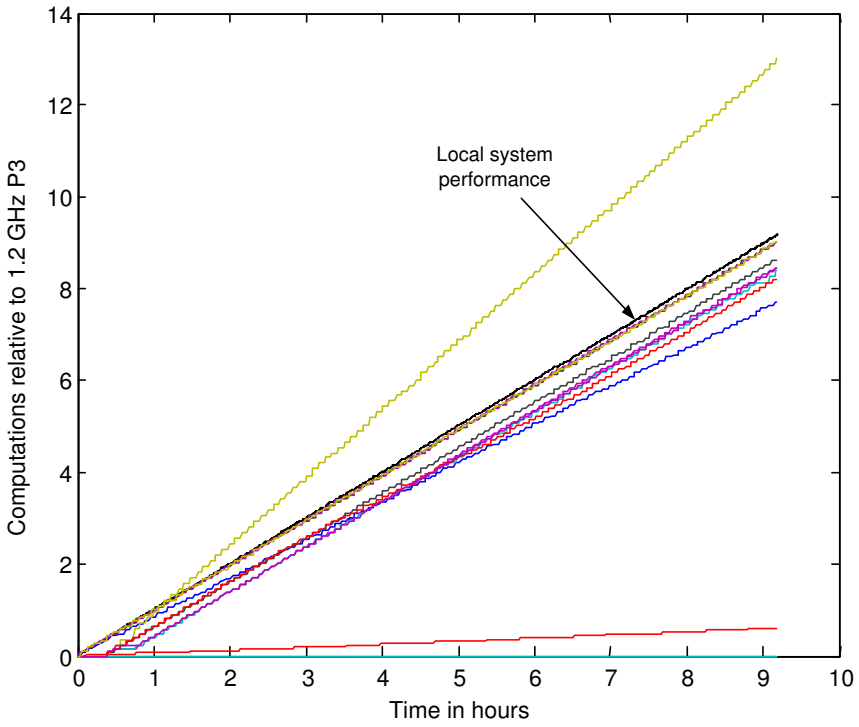

Figure 4.6: Performance plot-2, 11 tasks running in parallel, 1 was faster than the local machine (gold line), 9 were slightly slower and 1 was significantly slower (red line)

The bit error rates of IEEE 802.11a standard in AWGN and Rayleigh environments have been simulated using the Frontier platform as shown in Fig. 4.3 and Fig. 4.4. The parallelism in the simulation was achieved by simulating different data rates simultaneously and independently using the same executable. The executable needs to be compiled only once and thereafter when any changes are made to the Matlab programs. The executables were compiled and used for both windows and Linux platforms. Using the launch and listen strategy as mentioned in section 4.1.1 the application can be exited and also the machine can be shut down after the jobs are launched. At a later time, the application can be used to re-establish a connection with the Frontier Server and retrieve the results conveniently.

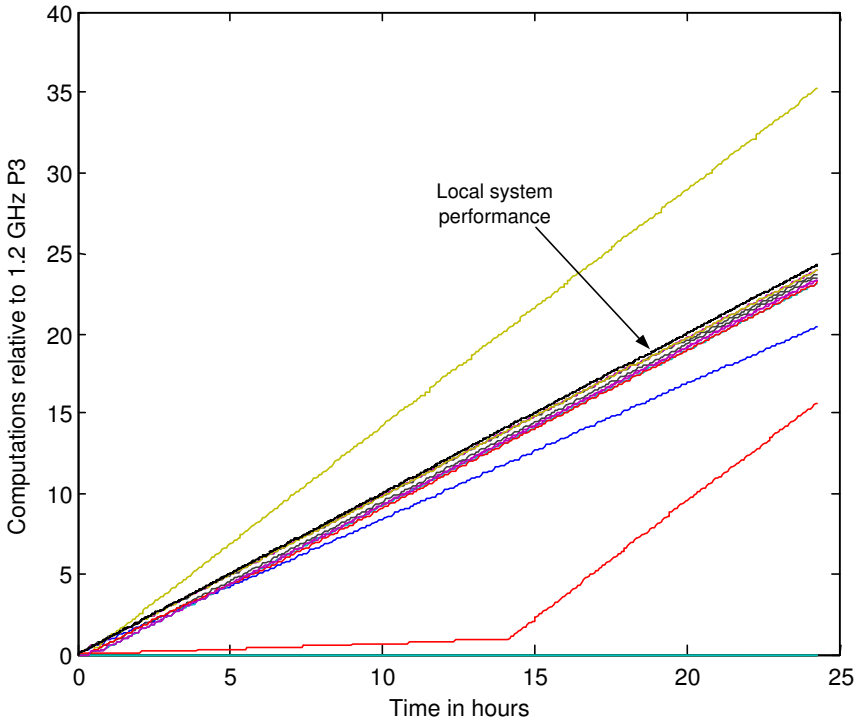

Figure 4.7: Performance plot-3, 11 tasks running in parallel After 24 hours, the grid executed 16,630,510 trials an order of magnitude improvement over running locally.

Chapter 5

Conclusion

5.1 Summary

In this thesis, a system model was presented and used to simulate the bit error rates of the IEEE 802.11a standard. A brief overview of grid computing was presented and desktop grids were explained in detail. A grid computing platform Frontier was used to develop an application to harness the idle CPU cycles of desktop resources. This application was used to efficiently perform simulation of BER for the physical layer of the IEEE 802.11a standard to very low values (10^{-7}).

5.2 Conclusion

The application developed was used to utilize the idle times of the desktop resources to perform simulations of communication systems. In addition, the performance of the grid was evaluated and results demonstrate that the grid performed the simulations much faster than that of a local machine. The speedup was achieved mainly due to two reasons, First Matlab programs were compiled into stand-alone applications that perform faster than Matlab programs since they execute without the need of a compiler. Secondly, simulations were executed in parallel by making use of the idle times of desktop resources already available. The processing power required to perform the simulations can also be satisfied by the use of a cluster computer. But, the advantage of using a desktop grid over a cluster computer

is scalability. In using a cluster computer the resources are limited by the size(number of processors) of the cluster whereas, in a desktop grid more number of nodes can be easily made available by connecting them to the grid without any significant amount of additional infrastructure costs .

5.3 Future work

The user interface of the application developed is Matlab. In turn developing a stand-alone executable of the Matlab programs used will completely obviate the need of Matlab. An improvement to the existing application would be developing a graphical user interface (GUI) to provide a better user-friendly environment. In the future, another interesting feature that would add a whole new dimension to the existing application would be to build a web-interface. It can enable the management of jobs from virtually anywhere by the user by signing into his account.

References

[1] B. Sklar, *Digital Communications*, 2nd ed., New Jersey, NJ: Prentice Hall PTR, 1995.

[2] "IEEE 802.11a Wireless LAN Medium Access Control and Physical Layer Specifications, High Speed Physical Layer in the 5Ghz Band," 1999.

[3] G. Caire and G. Taricco and E. Biglieri, "Bit-Interleaved coded modulation," *IEEE Trans. Inform. Theory*, vol. 44, no. 3, pp. 927 – 946, May 1998.

[4] A. J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Select. Areas Commun.*, vol. 16, no. 2, pp. 260 – 264, Feb. 1998.

[5] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 56–57, Jan. 1982.

[6] A. J. Viterbi, J. K. Wolf, E. Zehavi, and R. Padovani, "A pragmatic approach to trellis-coded modulation," *IEEE Commun. Mag.*, vol. 27, pp. 11 – 19, July 1989.

[7] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, July/Oct. 1948.

[8] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, New York, NY: John Wiley and Sons, Inc., 1991.

[9] J. Hagenauer, "Rate-compatible punctured convolutional codes (RCPC codes) and their applications," *IEEE Trans. Commun.*, vol. 36, no. 4, pp. 389 – 400, Apr. 1988.

[10] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. 13, pp. 260 – 269, Apr. 1967.

[11] I. Foster, "Internet computing and the emerging grid," *Nature, Macmillan Publishers Ltd Registered No. 785998 England*, Dec. 2000.

[12] J. F. Shoch and J. A. Hupp, "The worm programsearly experience with a distributed computation," *Communications of the ACM.*, vol. 25, pp. 172 – 180, March 1982.

[13] I. Foster and J. Geisler and B. Nickless and W. Smith and S. Tuecke, "Software infrastructure for the I-WAY high-performance distributed computing experiment," *High Performance Distributed Computing*, pp. 562 – 571, Aug. 1996.

[14] "The Globus project, www.globus.org," .

[15] A. J. G. Hey, "High performance computing-past, present and future," *Computing and Control Engineering Journal*, vol. 8, no. 1, pp. 33 – 42, Feb. 1997.

[16] S. J. Vaughan-Nichols, "New trends revive supercomputing industry," *Computer*, vol. 37, no. 2, pp. 10 – 13, Feb. 2004.

[17] I. Foster and A. Iamnitchi, "On death, taxes, and the convergence of peer-to-peer and grid computing," *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Feb. 2003.

[18] G. Flammia, "Peer-to-peer is not for everyone," *Intelligent Systems, IEEE [see also IEEE Intelligent Systems and Their Applications]*, vol. 16, no. 3, pp. 78 – 79, May-Jun 2001.

[19] A. Abbas, *Grid Computing: A Practical Guide to Technology and Applications*, Charles River Media, inc, 2004.

[20] I. Foster and C. Kesselman and J. Nick and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems Integration," *Open Grid Service Infrastructure WG, Globus Project*, June.

[21] I. Foster and C. Kesselman, *The Grid: Blueprint for a new computing infrastructure*, 2nd ed., Morgan Kaufmann, 1995.

[22] Parabon Inc., *The Frontier Application Programming Interface, Version 1.5.0.0011*, 2004.

A Design and User Notes

The software used to implement the grid computing platform discussed in this thesis was developed jointly with Jim O' Connor from Parabon Compuation [1]. This appendix contains design and user notes contributed by Jim O' Connor.

Design Overview:

The design of CMLGrid application consists of 4 major components:

1. A set of Matlab functions that are used to create and manage grid jobs from within Matlab. These are CmlGridSimulate, CmlGridList, CmlGridStatus, CmlGridSignal, CmlGridLog, CmlGridListen, and CmlGridRemove. Within Matlab you can type 'help CmlGrid' to get this list of commands. You can also view detailed help from within Matlab by typing 'help commandname'.

2. An Frontier application for launching the simulation job and listening to the results (GridSimulate.java). This application is launched using the GridSimulate.bat batch file. The command line syntax is:

GridSimulate.bat -mode (local — remote — launch — listen — status) jobid

Where the meaning of mode is:

local - run job on the local machine.
remote - launch the job on the grid and listen for results.
launch - launch the job then exit.
listen - listen to the specified job.
status - get the status for the specified job.

Note that the inputs for the job are contained in the job directory (see below). This input data is automatically generated by CmlGridSimulate.

Normally GridSimulate.bat is not called directly, but rather, is called from within the Matlab functions listed above.

3. Task code that runs on a grid node (SimulateTask.java).

4. A compiled version of the CML SingleSimulate function that performs the actual simulation. This is sent to the grid nodes as part of the task and is invoked by the task code

[1] www.parabon.com

(SimulateTask.java).

The basic flow of the application is that the user creates a grid job from within Matlab by calling CmlGridSimulate. This launches GridSimulate in remote mode and the application runs in background. The application listens for results and copies them to their final destination as they come in. The user can monitor and manage the job using the other CML grid functions. If for some reason the CmlGridSimulate application should terminate (e.g., the user shuts down the local machine), the job will continue running on the grid. The user can restart the listener using CmlGridListen command from within Matlab. It is important to note that all interaction with the grid job is done from within the Matlab environment.
Jobs Directory:

The state for grid jobs is stored in the <cml-home>/grid/jobs directory. The jobs directory has one subdirectory for each job. The name of this job directory is the job number. Each job directory contains the following sub-directories: input- Contains one SimSetupN.mat and SimStateN.mat file for each task N in the job. This is the input data for the task. targets- Contains one targetN.txt file for each task N in the job. This is the file path where the job's output should be written. log- Contains one taskN.out file for each task N in the job. This contains the stdout/stderr output from the task executable (SingleSimulate) as well as any errors thrown by the Frontier task code. This directory also contains job.out which contains the stdout/stderr of the GridSimulate application. signals- Used by CmlGridSignal to signal the listener. The following files may be written to this directory:

job.kill- The grid GridSimulate application periodically checks for the existence of this file, and if it finds it, the application removes the currently running job and exits. This is used by the CmlGridSignal command to terminate a job.

job.stop- The grid GridSimulate application periodically checks for the existence of this file, and if it finds it, the application exits without removing the job. This is used by the CmlGridSignal command to stop a listener.

N.kill- The grid GridSimulate application periodically checks for the existence of this file, and if it finds it, the application removes task number N. If task N is the last uncompleted task in the job, the job is removed from the server and the listener exits.

The job directory also contains several files that store job state:

jobinfo.mat- This is a MAT file containing jobid, number of tasks, and the simulation arguments. lockfile- This is a lockfile used by the GridSimulate application to determine if a listener is running. completed- This file is created by the GridSimulate application when the job completes. This is how CmlGridStatus tells if a job is complete or not.

Other Directories: bundles- This directory contains the native element bundles for the native elements used by GridSimulate. These bundles are created with the crbundle.bat script which is invoked from the Makefile.certs- This directory contains the Frontier credentials that will be used to run the job.conf- This directory contains Frontier configuration files. This directory also contains a subdirectory for each supported platform. The subdirectory contains a properties file that specifies platform specific properties for the task (task.properties).lib- This directory contains Frontier libraries (jar files).bin- This directory contains a subdirectory for each supported platform. The subdirectory contains the SingleSimulate executable, the SingleSimulate CTF file, and a platform- specific script that is used to launch the task on a grid node (startup.cmd).mat- This directory contains the CmlGrid Matlab script files. doc- This directory contains documentation. Specifically, it contains: README- A end-user README file. DesignNotes.txt- An overview of the CmlGrid design (this file). NativeCodeUserNotes.doc- A brief description of Frontier native data elements.

Other Files

crbundle.bat A windows bat file for creating native element bundles. crman.bat A windows bat file for creating native element manifests. GridSimulate.bat A windows bat used to launch the GridSimulate application. GridSimulate.java The java code for the GridSimulate application. SimulateTask.java The java code for the Frontier tasks launched by the GridSimulate application. Makefile Makefile to make the GridSimulate application and all its required elements.

GridSimulate and SimulateTask GridSimulate.java and SimulateTask.java make up the Frontier component of the CmlGrid application. GridSimulate.java is the code that runs on the local machine and SimulateTask.java is the code that runs on the grid node (and invokes the SingleSimulate executable). Before reviewing these files, it is strongly recommended that you read the Frontier Tutorial. This will give you a basic understanding of the structure of Frontier applications and make it much easier to understand the Frontier components of the

CmlGrid application.

This implementation makes heavy use of Frontier native data elements. See the file NativeCodeUserNotes.doc in this release for a description of using native data elements.

The basic design of the application is to create one task per SimInput/ SimState file in the input directory. SimInput and SimState are sent to the task as normal data elements. The SingleSimulate executable and ctf file are sent to the task as native data elements (because they are platform specific). Two other native data elements are sent to the task as well: startup.cmd and task.properties. The startup.cmd element is a platform-specific script used to launch the SingleSimulate executable. The task.properties file is a java properties file that is use to do platform specific initialization of the task.

When the task runs, it periodically sends back the SimState file, and the cmd.out file produced by the task (the report interval is controlled by the java property reportIntervalMin which is set in GridSimulate.bat). Locally these files are written to the result filename for the case and to the taskN.out file. When the GridSimulate executable terminates, the final versions of these files are returned.