Adaptive TCP Flow Control for Improving Performance of Mobile Cloud Clusters

by

Seungbae Lee

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 4, 2014

Keywords: mobile device, mobile ad hoc cloud, Hadoop mobile cluster, distributed computing performance, analytic data flow, TCP flow control

Approved by

Alvin Lim, *Chair*, Associate Professor of Computer Science and Software Engineering
David Umphress, Associate Professor of Computer Science and Software Engineering
Sanjeev Baskiyar, Associate Professor of Computer Science and Software Engineering
Xiao Qin, Associate Professor of Computer Science and Software Engineering

Abstract

Significant innovations in mobile technologies are enabling mobile users to make real-time actionable decisions based on balancing opportunities and risks in order to take co-ordinated actions with others in the workplace. This requires a new distributed analytic framework that collects relevant information from internal and external sources, performs real-time distributed analytics, and delivers a critical analysis to any user at any place in a given time frame through the use of mobile devices such as smartphones and tablets.

This work discusses the benefits and challenges of building mobile cloud clusters using recent mobile devices for distributed analytic applications by showing its feasibility with MapReduce framework and also investigates performance issues of Hadoop mobile clusters by conducting extensive experiments using typical Hadoop benchmarks; the newest release of Hadoop software framework with its enhancements is ported to the latest Android-based mobile devices through mobile virtualization technique. In addition, it develops the MapReduce simulator based on the ns-2 network simulator to comprehensively examine the performance and efficiency of mobile cloud clusters in extensive operating environments, which enables it to identify critical performance issues under different cluster (or workload) scales, dynamic node mobility, and various wireless channel conditions.

From the performance analysis this work identifies TCP (Transmission Control Protocol) communication problems resulting from distinct traffic patterns of MapReduce-based Hadoop distributed framework and proposes adaptive TCP flow control algorithms for improving the performance of mobile cloud clusters. The overall computing power of the mobile cluster is no longer significantly bounded by typical processing capabilities of each individual mobile node as mobile devices have been constantly enhanced, but the mobile cluster has limitations on interchanging large amounts of analytical data among mobile devices and

monitoring real-time status of cluster nodes through timely state updates, which result in significant delays in the processing time with corresponding performance degradation.

This work proposes an algorithm for cross-layer TCP flow control and dynamic network resource scheduling to avoid frequent overflows of the MAC (Media Access Control) layer transmit queue on the mobile nodes, which interrupt long-lived analytical data streams required for the partition and aggregation workflow of distributed analytic frameworks. It controls TCP's packet transmission based on the queueing level and implements dynamic resource scheduling for incoming and outgoing frames to minimize the queueing delay and stabilize the queueing level. In the evaluation test, the aggregate throughput of peer-to-peer TCP connections was significantly improved without incurring any throughput collapse.

In order to prevent TCP receive buffer overflows on the controller node due to the use of TCP push packets, which involve many interruptions in the reception of the latest status updates and progress reports from its worker nodes, this work also introduces another algorithm for mitigating the adverse effects of the TCP push flows that cause excessive transmissions from fast congestion window growth and frequent RTO (retransmission timeout) underestimation. It moderates the congestion window outburst and skips the RTO estimation using RTT (round-trip time) updates from the TCP push streams when the buffer overflow is detected. In the Hadoop TeraSort benchmark test, there were marked decreases in frequency of receive buffer overflows and TCP packet retransmissions and the overall processing time could be shortened accordingly.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many IT industry analysts predict that mobile growth is one of the most significant trends for the forthcoming years. In recent years, mobile technologies have matured and become suitable for a wider range of use, which creates new opportunities and demands for emerging platforms and services including mobile computing, hybrid cloud, and actionable analytics.

According to International Data Corporation (IDC) estimates, the recent surge in demand for mobile systems will continue to lead market growth of smartphones and tablet computers surpassing PC sales [1]. Gartner, Inc. also forecasts sales of 1.9 billion mobile phones in 2014, a 5 percent increase from 2013. In 2014, the worldwide tablet market is forecast to grow 47 percent with lower average selling prices attracting new users while world-wide shipments of traditional PCs are forecast to a 7 percent decline from 2013 [2].

The rising demand for mobile devices and mobility-related services has led smartphones and tablets to become far more powerful. Mobile devices with a quad-core 1.9 GHz processor and 2 GB memory are already widely available. Even octa-core processors and 3 GB memory modules for mobile devices are planned for release at the time of writing this dissertation. Along with enhancements in battery capacity and network capability, mobile devices are now capable of sharing their resources for distributed processing of critical data analytics as resource providers of cloud computing.

1.1 Recent mobile trends in the workplace

With the increasing popularity of mobile devices, a growing number of organizations are adopting Bring Your Own Devices (BYOD) programs [3]. With BYOD, workers bring

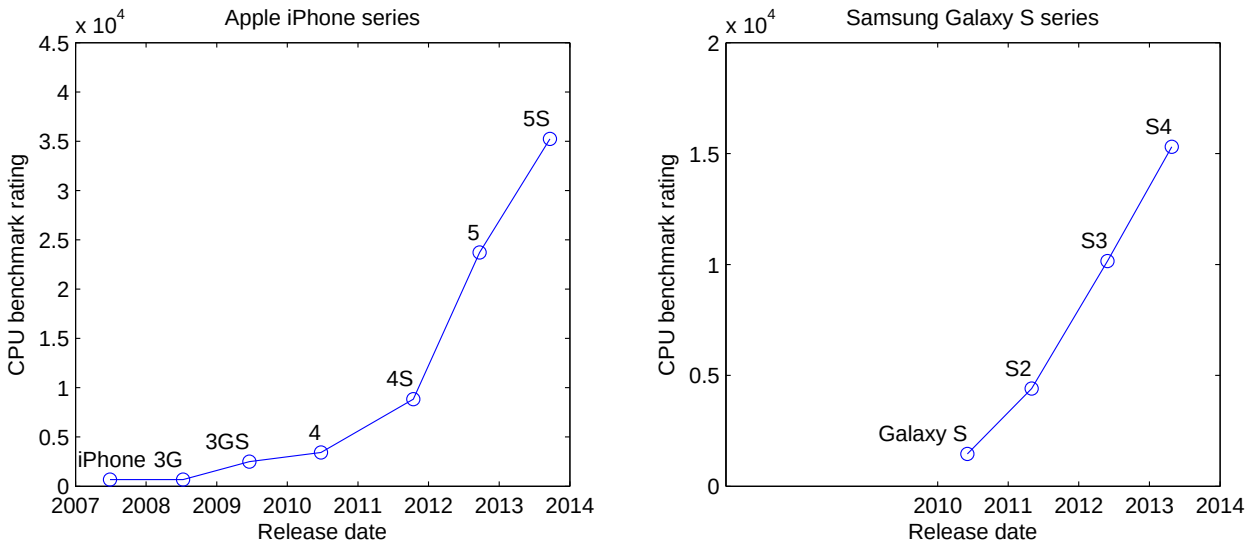


Figure 1.1: CPU performance improvement of Apple and Samsung's smartphones

their own mobile devices to their workplace and use those devices to access privileged information and run applications of their organization. It provides a great opportunity for improving productivity by accelerating the speed of decision-making and problem-solving and by allowing more flexibility.

1.1.1 Mobile device capabilities

The strong demand for diverse mobile devices and applications has led smartphones and tablets to offer the latest advanced features. Mobile platforms are leveraging multi-core processor architecture to dramatically increase processing power at a low cost. Manufacturers are introducing high-speed memory and increasing storage capacity for mobile devices. Moreover, advances in battery capacity and power saving techniques enable mobile devices to support large complex computations and long-running processes and provide more reliable high-speed wireless connectivity with more optional features, including 4G LTE, Wi-Fi, Bluetooth, and Near Field Communication (NFC). Figure 1.1 shows the CPU performance improvement (CPU benchmark results from PassMark [4]) of two flagship smartphones from

Table 1.1: Internal hardware sensors of Samsung Galaxy S4 smartphone

Sensors	Features
Accelerometer	Detects the mobile phone movement state based on three axes
Barometer	Identifies the atmospheric pressure at the user's current location
Geomagnetic sensor	Detects magnetic field intensity based on three axes
Gesture sensor	Recognizes the user's hand movements using infrared rays
Gyro sensor	Detects the mobile phone rotation state based on three axes
Hall sensor	Recognizes whether the cover is open or closed
Proximity sensor	Recognizes whether the mobile phone is located near the user by using infrared rays
RGB light sensor	Measures the red, greed, blue, and white intensity of the light source
Temperature sensor	Checks temperature level
Humidity sensor	Detects humidity level

Apple and Samsung Electronics, iPhone and Galaxy S series, which run mobile operating systems of Apple iOS and Google Android, respectively.

Furthermore, recent mobile devices provide innovative visual interaction through the use of advanced high-resolution touchscreens. Also, they integrate a variety of sensors that are constantly being improved, which include microphones, image sensors, 3-axis accelerometers, gyroscopes, atmosphere pressure sensors, digital compasses, optical proximity sensors, ambient light sensors, humidity sensors, touch sensors, etc, as detailed in Table 1.1. These cutting edge touchscreens and sensors enable mobile devices to monitor the operating environment in real time and adapt to the situation accordingly.

1.1.2 Mobility in the workplace

Today, people are connected in more ways than before. By using mobile devices, they no longer need to sit in front of desktop computers at office or at home in order to search for information and communicate with other people. People are creating new connections anywhere, anytime, and on any device. Mobility is having a huge impact on the way people work by saving time and resources and by opening new opportunities for innovation. A recent

IDC report shows the world's mobile worker population will reach 1.3 billion, representing 37.2% of the total workforce by 2015 [5]. The number of mobile workers in the U.S. will grow from 182.5 million in 2010 to more than 212.1 million by 2015.

The increasingly mobile and remote workforce is driving organizations to support a wide range of mobile applications and services, which enables workers to proactively detect and collect more information from internal and external sources by using mobile devices, and perform real-time analytics for rapid decision, thus improving collaboration and productivity. A Gartner's report shows that more than 40 percent of all enterprise web applications will support mobile environments by 2014 and 90 percent of companies will support corporate applications on mobile devices by 2014 [6].

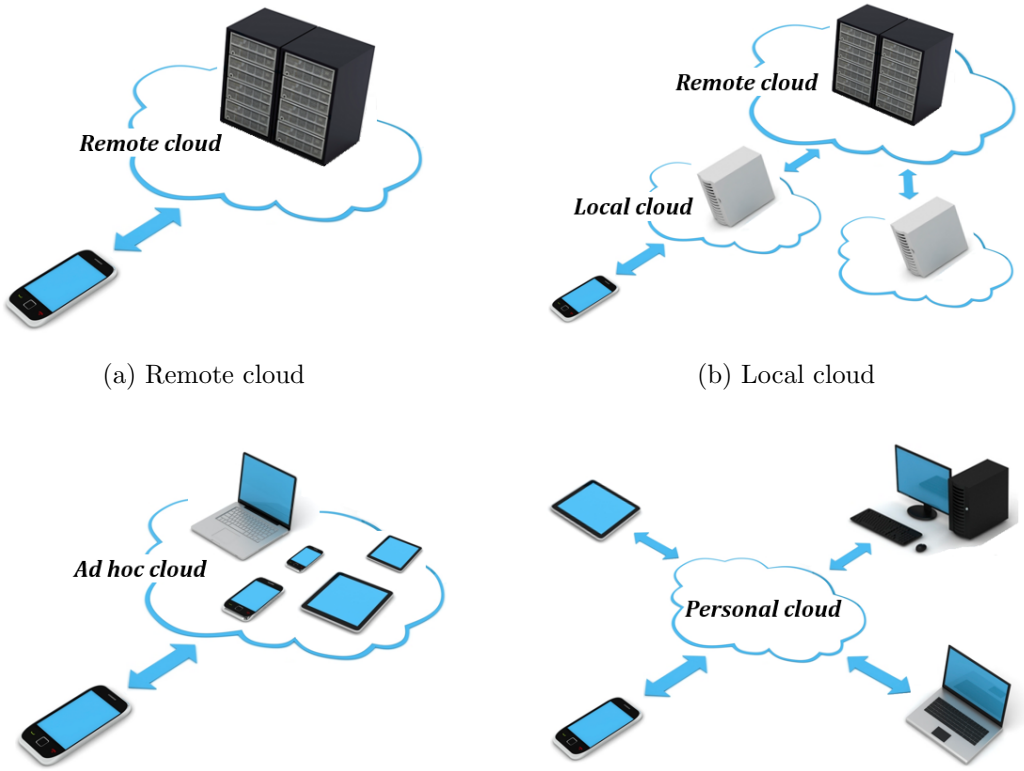
1.1.3 Actionable analytics

Conventional explanatory analytics usually focused on what happened in the past. Such analytics may be an outdated and ineffective approach to offering timely, accurate, and actionable insights needed for distributed decision making and coordinated action planning today. What is happening now? What is going to happen in the future? The ability to answer these questions in real time or near real time can provide a competitive advantage.

Recent advances in mobile technologies enable mobile users to collaborate with their team members through coordinated actions by balancing opportunities and risks. These actions can be generated by ad hoc distributed analytics that may consist of simulation, prediction, and/or optimization. This capability leads to a great opportunity for reducing cost while improving outcomes through more flexible decision-making that can be optimized for a specific scenario at a certain time and place [6].

1.2 Different types of mobile cloud computing

The increasing number of mobile applications require more complex processing and more operational data. These applications may include real-time mobile analytics that enhances



(a) Remote cloud

(b) Local cloud

(c) Ad hoc cloud

(d) Personal cloud

Figure 1.2: Different types of mobile cloud computing

situational awareness, risk assessment, distributed decision making, coordinated action planning, team collaboration, and instant responsiveness.

Despite the increasing use of mobile devices, exploiting its full potential is difficult due to the inherent problems such as resource limitations (e.g., low computational capability and battery capacity) and frequent disconnections from mobility. Mobile cloud computing can solve these problems cost-effectively by utilizing computing and storage capabilities from remote resource providers or other mobile devices. There are several approaches to mobile cloud computing with different concepts and configurations [7, 8]. Figure 1.2 illustrates the different types of mobile cloud computing.

1.2.1 Remote and local cloud services

The mobile cloud computing commonly means to run a mobile application (e.g., Google Maps) on a remote resource-rich server (e.g., Google Cloud Platform) while the mobile device acts like a thin-client connecting to the remote server through Wi-Fi or 3G/4G wireless networks. Mobile users can access cloud systems using a web browser or mobile application regardless of their location or device type. Other examples include mobile commerce (m-commerce), mobile learning (m-learning), mobile healthcare (m-health), etc.

Although the mobile cloud applications that connect to a remote infrastructure are becoming popular, they can perform well only under reliable connectivity. It is not practical to assume high-speed connections, seamless handovers, and fast responses on mobile devices as mobile environments are subject to high probability of significant disruptions to network services due to mobility, where fixed infrastructures are frequently unavailable and network partitions are common.

The Cloudlet proposed by [9] is another approach to mobile cloud computing. Mobile users usually run client-server software at a remote location and high network latency (or low network bandwidth) makes it insufficient for real-time applications. To cope with this issue, they introduced the concept of local cloud: trusted, resource-rich computers in the near vicinity of the mobile user (e.g., near or co-located with wireless access points). Mobile users can rapidly instantiate custom virtual machines (VMs) on the cloudlet running the required software in a thin-client fashion. The cloudlet can be deployed in common areas, such as public offices, airports, shopping malls, etc.

1.2.2 Ad hoc cloud and other concepts

Another approach is to consider other mobile devices as resource providers of cloud computing by making up a mobile ad hoc network as in [10]. Reliable access to remote resources is the first challenge in the mobile environments since the resources are commonly distributed across a variety of remote sources. Thus, clustering with nearby mobile devices

will promise faster connectivity and better availability. This work primarily focuses on this type of mobile cloud computing, where the remote resources are mobile and available only within the range of the wireless transmission.

The collective resources of the various mobile devices in a local area, and other stationary devices if available, are utilized for cloud computing in the mobile ad hoc cloud. As a result, it can effectively support user mobility, collective sensing, and distributed applications. On the other hand, a mobile device that initiates cloud computing in the ad hoc cloud needs to dynamically take advantage of mobile cloud resources depending on requirements of its workload because the ad hoc resources and operating environments of the mobile cloud are subject to change. Hence, monitoring and scheduling of available cloud resources are one of the most critical capabilities for the success of the mobile ad hoc cloud.

In addition to the approaches above, there are other concepts and models for implementing cloud services using mobile devices. For example, personal cloud and private cloud are operated solely for a single person and a single organization, respectively, public cloud is open for public use, and community cloud shares resources between several organizations from a specific community with common concerns [11]. Although there may be little or no difference between those cloud architecture, they have substantially different security consideration for resources and applications. Furthermore, Gartner identified hybrid cloud as one of the top strategic trends for 2014, which is composed of some combination of personal, private, public and community cloud services offering the benefits from multiple deployment models [12].

The rest of this paper is organized as follows. Chapter 2 introduces the background and work related to this work. Chapter 3 then describes the details of our experimental setup for mobile cloud clusters and conducts performance analysis to identify performance problems of mobile MapReduce applications. Chapter 4 presents problem statements and research questions of this study. Then Chapter 5 proposes adaptive TCP flow control algorithms for mobile clusters with an analysis of TCP performance problems and Chapter 6 provides

evaluation results for the proposed solutions. Finally, Chapter 7 concludes this paper with a summary of the previous chapters and a discussion of main contributions and further research ideas.

Chapter 2

Background and related work

This section provides background information about the distributed analytic framework for mobile cloud clusters and its communication protocol, and also presents summaries on previous studies related to this work.

2.1 Overview of Apache Hadoop for cloud clusters

When reviewing multiple distributed analytic frameworks, this work found that Apache Hadoop [13] can provide a good starting point for implementing mobile cloud clusters since it supports cost-effective and high performance analysis of a large volume of unstructured data on a set of commodity hardware. This section summarizes main ideas of MapReduce programming model and describe working mechanisms of Hadoop distributed file system and MapReduce task execution in Hadoop framework.

2.1.1 Apache Hadoop

Apache Hadoop is an open-source framework that uses a simple distributed processing model based on Google MapReduce [14] and Google file system (GFS) [15]. It effectively handles massive amount of information by either transforming it to a more useful structure and/or format, or extracting valuable analytics from it. Hadoop runs on any machines equipped with a lower cost processor and storage, and automatically recovers from hardware, and software failures by providing fault tolerance mechanisms. Hence, Hadoop is more cost-effective for handling large unstructured data sets than conventional data mining approaches by offering great scalability and high availability.

2.1.2 MapReduce programming model

MapReduce is a fundamental programming model in Hadoop architecture to process large volumes of data. MapReduce applications with a parallel approach utilize a scale-out architecture that makes use of inexpensive commodity servers configured as a cluster, which allows users to analyze terabytes or petabytes of data in a fast and reliable way. To take advantage of parallel processing that MapReduce provides, users need to submit a job with desired computations for a series of MapReduce tasks. MapReduce operates by dividing the processing into two phases: the Map phase and the Reduce phase. Each phase has key-value pairs as input and output, the data formats can be specified according to the application. The user implements two functions: the Map function and the Reduce function. The Map function reads each input key-value pairs and produces a list of intermediate key-value pairs. The Reduce function takes in all intermediate key-value pairs by sorting with the identical key and generates a final set of key-value pairs. Both the input and output key-value pairs in Map and Reduce phase are stored in an distributed file system.

The Hadoop platform that runs MapReduce applications automatically parallelizes the execution, coordinates network communication and ensures fault tolerance, which spares users from concerning about parallelization and hardware failures in a large-scale distributed environment. In addition, the Hadoop framework tries to assign Map and Reduce tasks to cluster nodes where the data to be processed is stored for alleviating loads on network bandwidth and preventing unnecessary network transfers, which is critical for high performance in data-intensive computing.

2.1.3 Hadoop distributed filesystem

Hadoop implements a distributed, scalable, and portable filesystem, called Hadoop distributed file system (HDFS), designed for storing large files with streaming access to data and optimized for data-intensive workloads such as MapReduce, running across clusters with commodity hardware. The data in the Hadoop cluster is broken down into small, fixed-size

pieces, called blocks; the default size of a block is 64 MB. The files in HDFS are split into block-sized chunks, which are stored as independent units. HDFS has two types of instance in a master-slave pattern: a single NameNode (the master) and a number of DataNodes (the slaves). NameNode manages the filesystem namespace which maintains the filesystem tree and the metadata for all the files and directories in the tree. NameNode also monitors the DataNodes on which all the blocks for a given file are located. DataNodes are the worker nodes of the filesystem. They store and retrieve blocks at the request of clients or NameNode through remote procedure call (RPC), and report their recent status updates periodically to NameNode with the metadata of their current blocks.

When an HDFS client wants to read a file, it first contacts NameNode to obtain the location information of data blocks comprising the file and then retrieves block contents from the DataNodes closest to the client. When writing data, the client also requests NameNode to schedule a set of DataNodes to store the block replicas for fault tolerance and then writes data to the DataNodes in a pipeline-like fashion. Since the current architecture has a single NameNode for each cluster where NameNode keeps all the namespace and block locations in memory, the size of the NameNode's heap limits the number of files as well as the number of blocks addressable. This also limits the total cluster storage that can be supported by NameNode.

2.1.4 MapReduce task execution

A typical Hadoop cluster consists of a single master node and many slave nodes. An entire MapReduce computation is called a job and the execution of a Map or Reduce function on a slave is called a task. A sever process running on the master node, named JobTracker, coordinates jobs on a cluster of slave nodes. The JobTracker is responsible for accepting jobs from clients, dividing the input data (i.e., file) into many splits (i.e., blocks) and assigning the blocks to Map tasks that are processed concurrently on multiple slave nodes. Each slave node runs a client process called TaskTracker that manages the execution of the tasks

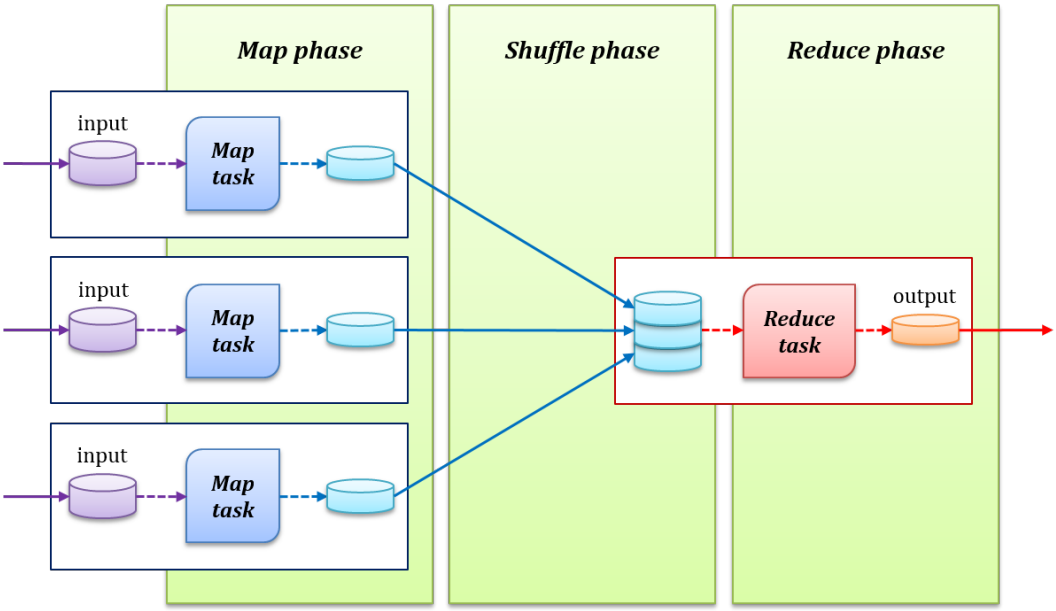


Figure 2.1: Data flows in MapReduce task execution

assigned to the node. Each TaskTracker has a fixed number of slots for executing tasks; the number of slots on each cluster node may be different and it depends on the node's hardware capacity.

After the MapReduce job is submitted, the number of Map tasks is determined by the total number of data blocks spilt from input files. Each Map task reads a block from HDFS, parses it into key-value pairs and applies the Map function to each pair. Once the intermediate key-value pairs are generated on the local disk by the Map tasks, partitions on the intermediate pairs are performed using a scheme that keeps pairs with the identical key to be processed on the same Reduce task. After that, JobTracker sends the locations of the pairs to corresponding Reduce tasks. MapReduce guarantees that the intermediate result to every Reduce task is sorted by key. The process by which the system performs the sort and transfers the Map outputs to the Reduce tasks as inputs is known as the shuffle. After a Reduce task has read all the intermediate pairs, it runs the Reduce function and writes the output pairs to a final output file in HDFS. The data flows in MapReduce execution consisting of the Map, Shuffle, and Reduce phase are illustrated in Figure 2.1.

This section omits some details of configuration parameters, performance metrics, and analytic algorithms in Hadoop architecture, which can be found in many excellent publications including tutorials, academic papers, and technical reports. Some of them (e.g., [16], [17], [18], and [19]) are referenced in this work.

2.2 Overview of TCP flow control for performance analysis

Most of the current distributed systems including Hadoop analytic framework extensively employ Transmission Control Protocol (TCP) for reliable communications between cluster nodes. Since TCP is optimized for accurate delivery rather than timely delivery, TCP on mobile devices may frequently incur significant delays while waiting for retransmissions of lost or damaged packets and rearrangements of out-of-order packets on wireless communication links as detailed in [20]. This section presents an overview of TCP flow control mechanisms to provide a background for performance analysis of mobile cloud clusters.

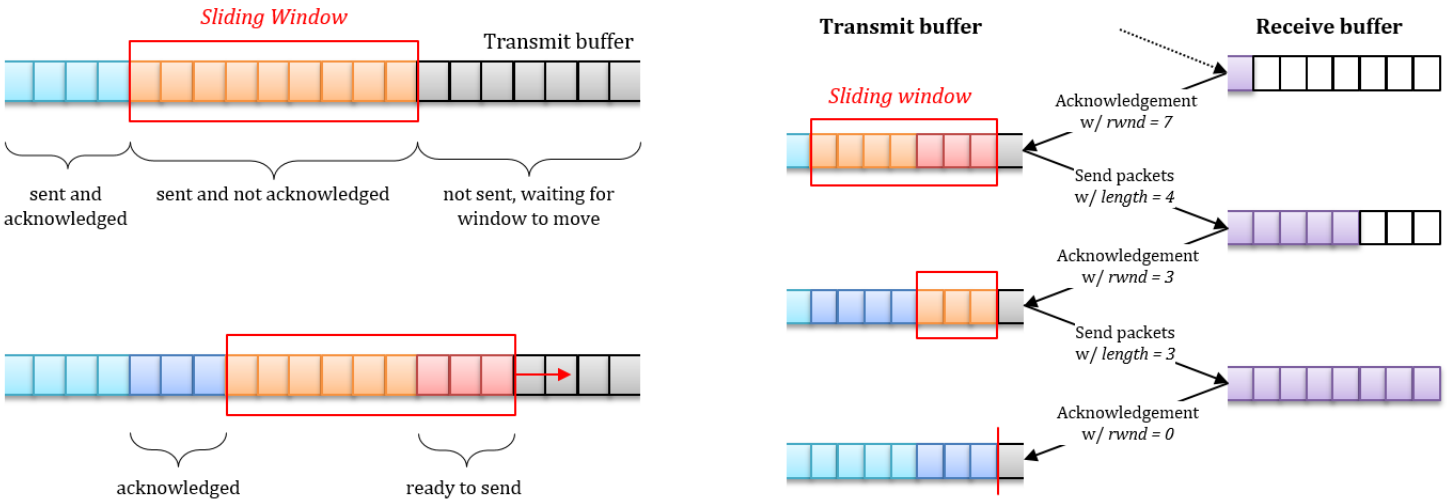
2.2.1 Sliding window flow control

TCP provides reliable data transfer with its flow control and congestion control mechanisms between applications on two hosts in the network. Since the TCP sender that desires to transmit a large amount of data may try to send data too fast for the TCP receiver to receive and process it reliably, the TCP standard [21] describes a sliding window based flow control mechanism. The TCP sender first buffers all data (in a TCP transmit buffer) before the transmission, assigning a sequence number to each buffered byte. Pieces of the buffered data are continuously formed into TCP packets that include a sequence number of the first data byte in the packet. Then a portion (window) of the packets that are ready to send is transmitted to the receiver using the IP protocol. As soon as the sender receives delivery confirmation for at least one transmitted packet, it transmits a new portion of packets; the window slides along the sender's buffer as shown in Figure 2.2a. Since packet transfers over bad network conditions may not be reliable, the receiver is required to respond with an

acknowledgment as it receives the data while the sender keeps a record of each packet it transmits. The sender also maintains a retransmission timer, and retransmits a packet if the timer expires before the data has been accurately acknowledged.

Although a sliding window based flow control is relatively simple, it has conflicting objectives. In order to maximize the throughput of a TCP flow, it is essentially required that the size of a sliding window also be maximized. On the other hand, if the sliding window is too large, there is a high probability of packet loss because the network and the receiver have resource limitations. Thus, reducing packet losses requires minimizing the sliding window. Hence, finding an optimal value for the sliding window parameter (which is usually referred to as the congestion window size) that provides better throughput, yet does not overwhelm the network bandwidth and the receiver capacity is a critical problem in TCP communications.

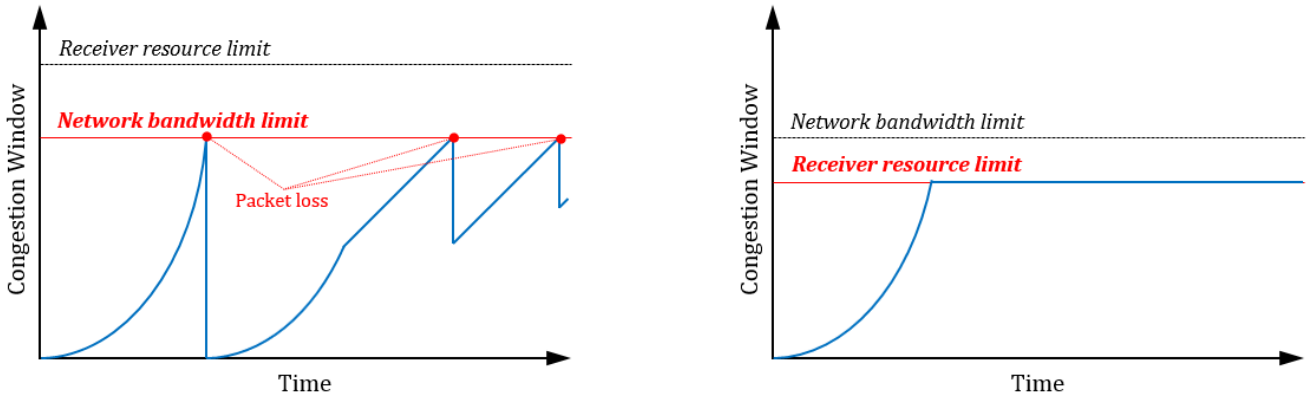
The TCP flow control mechanism integrates the receive window concept as well, which is designed for the receiver to share the information about the available receive buffer with the sender. In each TCP segment, the receiver specifies the amount of additionally acceptable data (in bytes) in the receive window field. The sender can send only up to that amount of data before it must wait for an acknowledgment and window update from the receiver. Figure 2.2b illustrates this mechanism. When establishing a connection, the receiver informs the sender about the available buffer size for incoming packets (in the example shown, the receiver's window reported initially is 7). The sender transmits a portion (window) of data packets. This portion must not exceed the receiver's window and may be smaller if the sender is not willing (or ready) to send a larger portion. In the case where the receiver is unable to process data as fast as the sender generates it, the receiver reports decreasing values of the window (3 and 0 in the example). This induces the sender to shrink the sliding window. As a result, the TCP transmission will eventually synchronize with the receiver's processing rate as demonstrated in Figure 2.3.



(a) Sliding window control

(b) Receive window control

Figure 2.2: Sliding window based TCP data transfer process



(a) Network limit

(b) Receiver limit

Figure 2.3: Upper bound of TCP congestion window growth

When a receiver advertises a window size of 0, Zero Window (i.e., a receiver is not able to receive any more data at the moment) as shown in Figure 2.2b, the sender stops transmitting data and starts a persist timer. The persist timer protects TCP from the deadlock situation that the sender will never be able to transmit further data by waiting for a new window update from the receiver, which could arise if a subsequent window update from the receiver is lost. When the persist timer expires, the TCP sender attempts recovery by transmitting a small packet (called Zero Window probe) so that the receiver forcibly responds by sending another acknowledgement containing the receive window update.

2.2.2 Packet loss detection mechanisms

The early and accurate detection of packet loss is one of core mechanisms in the TCP congestion and flow control. Most of the TCP control proposals gradually increase the utilization of network resources up to the limit where a packet loss is starting to be detected, at which point they reduce its transmitting rate, retransmit the lost packet, and begin another phase of rate controls (e.g., rate increases). TCP detects a packet loss through two critical mechanisms: retransmission timeout and duplicate acknowledgement [21].

The first mechanism defines the retransmit timeout (RTO), in which TCP waits for a timeout of the retransmission timer for the detection of a packet loss. Although it is capable of reliably detecting all losses, the detection is not fast enough and is necessarily time consuming since the retransmission timer must be set high enough to avoid unnecessary timeouts caused by transient network conditions; the minimum time when a loss can be detected is the round-trip time (RTT) and the RTO should be greater than the RTT.

If the RTO value is overestimated, the TCP packet loss detection mechanism becomes very conservative, and performance of individual flows may be significantly degrade. In the opposite case, when the RTO value is underestimated, the loss detection mechanism may cause unnecessary retransmissions by wasting the available network resources and aggravating the network congestion level. For the problems of erroneous RTO estimates, the RTT

variance estimation algorithm [22] tries to alleviate the overestimation problem by establishing a fine-grained upper bound for the RTO and the exponential RTO backoff algorithm [23] to mitigate the underestimation problem by doubling the RTO value on each retransmission occurrence.

The second mechanism assumes that if TCP receives a few duplicate acknowledgements of a packet then the packet was lost, in which TCP counts the number of acknowledgements with the same sequence number for the detection of a packet loss. When the probability of packet duplication (and packet reordering) in the network is negligible, the duplicate acknowledgments can be considered a reliable loss indication. Thus, the sender can retransmit a lost packet faster without waiting for the corresponding retransmission timer to expire.

2.2.3 Congestion window dynamics

The initial TCP standard lacks any means to adjust the transmission rate to the actual capacity of the network. When a TCP sender (or many senders) is transmitting too much data that can exceed the available network bandwidth, congestion collapse easily occurs in the form of queueing delay, packet loss or the blocking of new connections, which result in an substantial reduction in network throughput. To resolve this congestion collapse problem, a number of solutions have been proposed. Most of them share the same idea of utilizing a network-aware rate limiting mechanism along with the receiver-based flow control. For this purpose the congestion window mechanism was introduced, in which the TCP sender determines (or estimates) the number of data packets that the network can accept for delivery without becoming overloaded. If the receiver does not have any resource limitations, the congestion window limit can be considered an indication of the maximum capacity of the connection. On the other hand, when the flow control limit (i.e., the receive window) is less than the congestion control limit (i.e., the congestion window), the former is considered a real bound for outstanding data packets. Graphs in Figure 2.3 show two cases of the congestion window dynamics; the left graph shows the congestion window dynamics when the network

cannot deliver any more data at the transmitting rate, and the right graph represents the case when the receiver cannot process further data at the receiving rate [20].

One of the earliest solutions to solve the congestion problem in TCP operation has been proposed by [22]. The solution is based on the original TCP specification and includes a number of algorithms to avoid the network congestion. The most important algorithms are the Slow Start and Congestion Avoidance algorithm. These provide two slightly different distributed peer-to-peer mechanisms which allow a TCP sender to detect available network resources and adjust the transmission rate of the TCP flow to the limit determined. Assuming the probability of random packet corruption during transmission is negligible, the sender can consider all the packet losses as the congestion problem. Furthermore, the reception of any acknowledgement packet is an indication that the network can accept and deliver a new packet. Thus, the sender expecting that it will not result in any congestion can transmit at least the amount of data that has just been acknowledged. This incoming and outgoing packet balancing is called the packet conservation principle that is the basic concept of both Slow Start and Congestion Avoidance algorithm.

In the Slow Start algorithm, the reception of an acknowledgement packet indicates availability of network resources for transmitting double the amount of data (i.e., multiplicative increase) that has been acknowledged by the receiver. In other words, instead of a linear growth in the congestion window, its growth follows an exponential function (i.e., the growth is quite aggressive). If a packet loss is detected (i.e., the network is experiencing congestion because all network resources have been utilized), on the other hand, the congestion window is reset to the initial value (e.g., the maximum segment size) to ensure release of network resources. The other algorithm for improving TCP effectiveness in the networks with resource limitations is Congestion Avoidance that combines linear growth of the congestion window with an exponential reduction when a congestion takes place. In comparison to the Slow Start, this algorithm is much more conservative in response to acknowledgement of transmitted packets and to detection of packet losses. As opposed to doubling the size, the

congestion window increases by one (i.e., additive increase) only if all data packets have been successfully delivered during the last RTT. After a loss is detected, the algorithm reacts in a different way, which cuts the congestion window size by half (i.e., multiplicative decrease). Figure 2.3a demonstrates the behaviors of the Slow Start and Congestion Avoidance algorithm over the bandwidth-limited network.

A number of proposals to improve various aspects of TCP performance (e.g., the effective use of available network resources) have been presented over the past 20 years, which include mechanisms to probe the available network resources, estimate the network congestion state, and detect the packet loss under different assumptions on the network environment. Moreover, some proposals contain algorithms to improve the poor utilization of error-prone wireless and high-speed wired networks. For example, TCP Westwood+ [24] proposes a bandwidth estimation technique for wireless networks with random loss and TCP CUBIC [25]) a congestion control algorithm that scales well in high-speed networks with long latency. The current Linux kernel supports some of those proposals and users can choose a better algorithm for a particular network connection; TCP CUBIC is implemented by default in Linux kernel 2.6.19 and above. However, there are not yet the practical guideline and performance criteria for the selection of a congestion control algorithm.

2.3 Related studies on mobile cloud clusters

Many researchers have identified key attributes, technologies, and challenges that distinguish cloud computing from traditional computing paradigms [26, 27, 28, 29, 30]. To put it briefly, cloud computing provides reliable, customizable and dynamic computing environments with Quality of Service (QoS) guarantee for end-users [31]. Also, many studies have been interested in mobile cloud services on the Internet as summarized in [7] and [8].

This work pays particular attention to the performance of mobile ad hoc cloud, where ad hoc networks of mobile devices themselves work as resource providers of the cloud. In

this type of cloud, the workload and data reside on individual mobile devices rather than on remote resource servers.

2.3.1 Implementation of mobile ad hoc cloud

Hyrax [32, 33] explores the feasibility of using a cluster of mobile phones as resource providers by porting Apache Hadoop to Android smartphones. For a sample application, they present a simple distributed mobile multimedia search and sharing program. However, their performance evaluations for the Hadoop mobile cluster are limited since they completed only a partial implementation of the Hadoop architecture, where many core features had to be removed due to difficulties and obstacles in Hadoop migration. Even the major controllers of Hadoop framework, such as JobTracker for MapReduce and NameNode for HDFS, are not installed on the mobile node. A similar approach to implementing the Hadoop framework on mobile devices is found in [34].

Serendipity [35, 36] discusses the challenges of remote computing using mobile devices and introduces a framework that enables a mobile computation initiator to use remote computational resources available on mobile devices. They implement an actual prototype on Android devices and evaluate their system using two sample applications: a face detection application and a speech-to-text application. However, no performance comparison with the existing distributed frameworks is made. Another study, Cuckoo [37], proposes a computation offloading framework for Android smartphones and illustrates its utility with an application for multimedia content analysis.

In short, several studies on the ad hoc cloud framework for mobile devices have been conducted by implementing only part of an existing distributed analytic framework or by proposing a customized framework similar to the existing one [38]. Furthermore, the previous studies are mostly evaluated using just one or two domain-specific applications and fail to provide comparative analysis of their performance and efficiency. To the best of our

knowledge, there has been no comparable framework and performance analysis for practical mobile cloud clusters running distributed analytic applications.

Although this work mostly focuses on the performance of practical distributed analytics on the mobile cloud clusters in terms of job processing time and response time, other work concentrates on mobile device's energy efficiency which is a key aspect to enable data analysis and mining over mobile devices. For example, an energy-aware scheduling over the mobile cluster to optimize energy utilization should be taken into account for mobile distributed analytics.

2.3.2 Network problems of traditional cloud

This work also focuses on reliable data communications between mobile devices for analytical data transfers in the workflow of distributed analytics under the limitations of TCP performance over wireless links. To find the best way to control data flows on mobile devices for improving performance of mobile cloud clusters, it is necessary to review previous solutions for the typical datacenter cloud.

In datacenter networks, there have been many proposals to solve typical network problems in many-to-one communication patterns, known as TCP Incast, where the traffic bursts overload the switch buffers, which lead to a significant increase in queueing delay and decrease in TCP throughput. The problems affect the performance of cloud computing frameworks in which distributed processing cannot progress until all parallel threads in a stage complete [39]. Examples of such frameworks include distributed filesystems, web search, and other applications with partition or aggregation workflows [40, 41, 42].

The traditional solutions to TCP Incast include modifying network parameters [41, 43] or TCP congestion/flow control algorithms in Link and Transport network layer [44, 45], which may involve customized network designs for the efficient switching [40, 42], and optimizing application data transfer patterns for mitigating the TCP congestion [46, 42]. Since TCP Incast is originally incurred in the switch-based network topology using TCP,

however, most of the solutions may be inapplicable directly to the mobile network issues that arise in the wireless environments with different characteristics.

2.3.3 MapReduce performance prediction using simulation

Simulation has been widely used for performance prediction and characterization, which can simplify research process by skipping intricate processes of experimental setup and configuration and significantly reduce the experiment time. Since MapReduce has been adopted as a preferred choice of framework for data intensive computing, there have been many efforts toward developing MapReduce simulators over the past few years to address the performance analysis of scheduling algorithms. They provide several MapReduce simulators such as Mumak [47], MRSim [48], and MRPerf [49].

Mumak uses job trace of real world workload as input to estimate job execution time, throughput, etc. MRSim is another simulator based on discrete event simulation, which can predict job performance as well as hardware utilization. The other simulator is MRPerf that uses information about node specification, cluster topology, data layout, and job description as inputs and generates a detailed phase-level execution trace that provides job execution time, amount of data transferred, and time-line of each phase of the job execution [19].

MRPerf is uniquely based on the popular ns-2 network simulator [50], which models task communications over the networks with varying cluster configurations and different network topologies to simulate correct network behavior of the real system. Although this work primarily performs actual experiments using Hadoop clusters consisting of practical mobile devices to identify performance issues of mobile cloud clusters, it also develops a MapReduce simulator based on an existing simulator (MRPerf on ns-2) to address dynamic node mobility under various wireless environments and capture more details of performance aspects.

Chapter 3

Understanding performance issues of Hadoop mobile clusters

MapReduce is the fundamental software programming model in the Hadoop architecture, which performs distributed processing of large data sets on a computing cluster. A single large workload (job) is divided or mapped into smaller sub-workloads (tasks) to be processed in parallel. The results from the sub-workloads are merged, condensed, and reduced to produce the final result. Both input and output are stored on the nodes throughout the cluster in the distributed filesystem.

Numerous factors can affect the performance of the Hadoop cluster. The typical performance factors such as workload type, cluster size, input/output data size, and characteristics of computing nodes (e.g., CPU, memory, and I/O resources) have significant impacts on the processing time. In addition, the network is also a critical factor on the Hadoop performance since the nodes are interconnected through the network in order to transfer data for distributed processing during one or more phases of MapReduce execution.

This chapter examines the performance of MapReduce in practical mobile cluster setups using Hadoop benchmarks and identifies its critical performance issues in the mobile operating environments.

3.1 Hadoop benchmarks for performance evaluation

A benchmark provides a method of comparing the performance of various subsystems across different system architectures and mimics a particular type of workload on a component or system. In order to perform extensive performance analysis of Hadoop mobile clusters, this work chooses workloads from a benchmark suite provided by [51]. It contains

typical Hadoop workloads with options for input/output configurations (e.g., data size, compression method, etc). Table 3.1 lists its benchmark workloads with their characteristics. All the workloads are implemented with Hadoop MapReduce framework and are capable of performing a variety of data intensive computations such as sorting, I/O operations, web search and machine learning.

The Sort, WordCount and TeraSort benchmark for micro-benchmarking are currently implemented in Hadoop software releases, which are three most popular benchmarks widely used by Hadoop developers and engineers. The Sort and WordCount are representative of a large group of real world MapReduce applications that extract the interesting information from large input data. The Sort benchmark simply uses the MapReduce framework to sort the input directory into the output directory, where the inputs and outputs must be sequence files with keys and values. The WordCount benchmark reads text files and counts how often words occur. Each Map task takes a line as input, breaks it into words and then generates a key-value pair of $< word, 1 >$. The Reduce task sums the counts for each word and generates a single key-value. The TeraSort benchmark sorts data at terabyte scale, which has been used in many cluster competitions among distributed computing platforms to show their performance and efficiency.

The TestDFSIO benchmark also included in the Hadoop releases is another microbenchmark for I/O performance, which performs parallel file read and write operations in separate Map tasks. The output of the Map task is used for collecting statistics relating to the file just processed. The statistics are accumulated in a Reduce task to generate a summary on average I/O throughput of the distributed file system (i.e., HDFS).

The Nutch Indexing and PageRank benchmark represent the web search applications, which are open-source applications. The Nutch Indexing benchmark that comes from Apache Nutch project crawls web links from root URLs and converts the link information into inverted index files with MapReduce tasks. The PageRank benchmark is composed of a

Table 3.1: Hadoop benchmarks and their characteristics

Category	Workload	Resource utilization	Data flow patterns
Microbenchmarks			
	Sort	I/O bound	$\xrightarrow{\textbf{\textit{data}}}$ Map $\xrightarrow{\textbf{\textit{data}}}$ Reduce $\xrightarrow{\textbf{\textit{data}}}$
	WordCount	CPU bound	$\xrightarrow{\textbf{\textit{data}}}$ Map $\xrightarrow{\textbf{\textit{data}}}$ Reduce $\xrightarrow{data}$
	TeraSort	Map: CPU bound, Reduce: I/O bound	$\xrightarrow{\textbf{\textit{data}}}$ Map $\xrightarrow{\textbf{\textit{data}}}$ Reduce $\xrightarrow{\textbf{\textit{data}}}$
	TestDFSIO	I/O bound	Map $\xrightarrow{data}$ Reduce
Applications			
Web search	**Nutch indexing**	Map: CPU and I/O bound, Reduce: I/O bound	$\xrightarrow{\textbf{\textit{data}}}$ Map $\xrightarrow{\textbf{\textit{data}}}$ Reduce $\xrightarrow{data}$
	Page rank	CPU bound	$\xrightarrow{\textbf{\textit{data}}}$ Map $\xrightarrow{data}$ Reduce $\xrightarrow{data}$
Machine learning	**K-mean clustering**	I/O bound,	$\xrightarrow{\textbf{\textit{data}}}$ Map $\xrightarrow{data}$ Reduce $\xrightarrow{data}$
		1^{st} Map: CPU bound	$\xrightarrow{\textbf{\textit{data}}}$ Map $\xrightarrow{\textbf{\textit{data}}}$
	Bayesian classification	CPU bound in iteration,	$\xrightarrow{data}$ Map $\xrightarrow{\textbf{\textit{data}}}$ Reduce $\xrightarrow{\textbf{\textit{data}}}$
		I/O bound in clustering	$\xrightarrow{\textbf{\textit{data}}}$ Map $\xrightarrow{\textbf{\textit{data}}}$ Reduce $\xrightarrow{data}$

Note: Font size denotes the data size of the input, intermediate, and output; $\xrightarrow{\textbf{\textit{data}}} >> \xrightarrow{\textbf{\textit{data}}} >> \xrightarrow{data}$

chain of Hadoop jobs calculating the rank of each web page according to the number of reference links.

The K-means Clustering and Bayesian Classification are also introduced as the machine learning applications. The K-means clustering benchmark takes in a numerical vector in n dimensional space representing the features of the objects to be clustered. The algorithm that randomly chooses k points in the vector space serving as the initial centers of the clusters recalculates the center of each cluster iteratively with MapReduce jobs until the points are not reselected or the maximum number of iterations is reached. The Bayesian Classification benchmark involves four chained MapReduce jobs that extract labels from input text, compute the Term Frequency-Inverse Document Frequency (TFIDF) of each feature in each label, and then perform the weighting and normalization.

3.2 Assumptions on mobile cloud clusters

This work performs actual experiments and runs simulations based on the following assumptions on the basic, common configurations of practical mobile cloud clusters for ad hoc analytics. However, future work will consider extensive scenarios that include dynamic node mobility and complicated analytical workloads over various mobile environments.

- Mobile devices may process computational workload that exceeds their capability by offloading portions of the workload to remote resources for distributed execution. All mobile devices are capable of sharing their computing resources, and behave in a collaborative and trustworthy manner.

- Clustering with nearby mobile devices to build a mobile ad hoc cloud provides faster connectivity and better availability because the actual connectivity with typical remote cloud infrastructures may be intermittent and unpredictable due to the mobility of mobile devices.

- Mobile nodes belonging to a cluster only communicate with adjacent nodes within their communication range in a wireless single-hop network. The data transmission of the multiple nodes might interfere with each other due to the overlapping communication ranges on a shared channel.

- The mobile cluster runs a single workload at a time, either transforming the unstructured input data to a more useful structure without adding new data, or extracting small but valuable analytics from the input data. The amount of intermediate and output data generated by mobile devices depends on the type of workload.

3.3 Performance experiments of Hadoop mobile clusters

This section describes details of the practical experimental setup for Hadoop mobile clusters and presents the experimental results for performance analysis.

3.3.1 Experimental setup

In the experiments this work measured the performance of Hadoop clusters using Android-based mobile platforms including smartphones (e.g., Samsung Galaxy S2 and Google Galaxy NEXUS), media players (e.g., Samsung Galaxy player), and tablets (e.g., Samsung Galaxy Tab and Google NEXUS 7) under extensive distributed configurations. This chapter presents experimental results from one of those cluster setups, which consists of eleven NEXUS 7 tablets developed by Google in conjunction with Asus. Figure 3.1 displays the experimental mobile cluster with Google NEXUS 7 tablets.

The experimental platform, NEXUS 7, is the first tablet in the Google Nexus series that implements the Android operating system. The Nexus 7 features a 7-inch display, NVIDIA Tegra 3 quad-core processor, 1 GB of memory, and 16 GB of internal storage, and incorporates built-in Wi-Fi, Bluetooth, and NFC connectivity [52]. The tablet runs the latest Android operating system (version 4.2.2, nicknamed Jelly Bean) and Hadoop stable release (version 1.1.2) with Oracle JDK (version 1.6) at the time of writing this paper. The

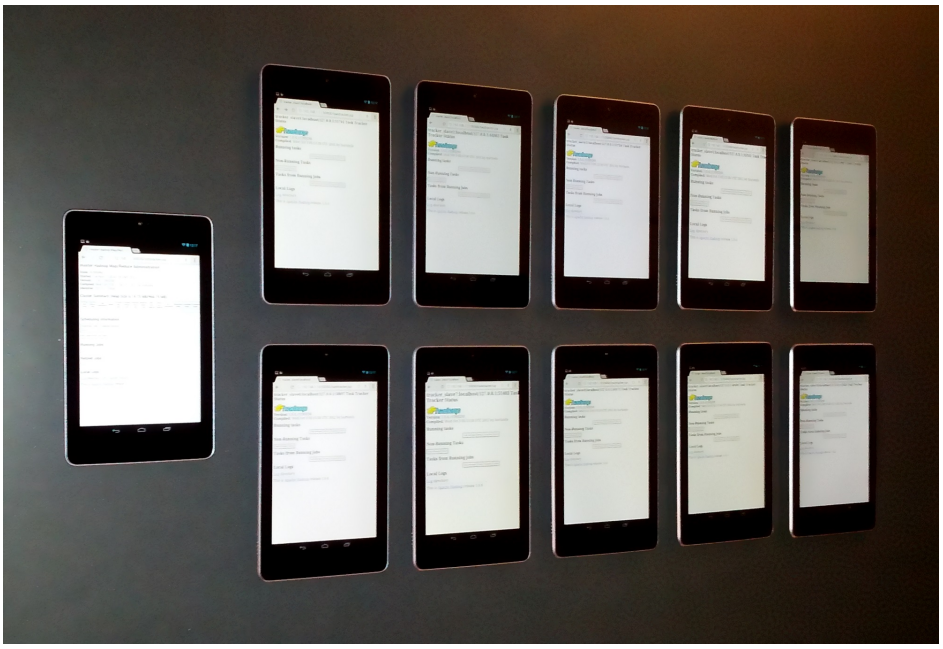


Figure 3.1: Experimental mobile cluster using Google NEXUS 7

detailed specifications of experimental platforms are listed in Table 3.2. All platforms are reliably interconnected with a Wi-Fi based wireless access point, Asus RT-N66U, in an IEEE 802.11n [53] infrastructure mode.

Android is a mobile operating system designed for smartphones and tablets, which makes use of a virtual machine on a customized embedded Linux system as its runtime environment to run mobile applications. The virtual machine provides an isolated environment for code execution, where an application with a malicious piece of code cannot directly affect the system (i.e., the core OS will be kept from getting corrupted). Thus, it makes the system more stable and reliable. In addition, it allows for cross-compatibility as its applications can be executed on any mobile platform using the virtual machine. The android applications are usually written in Java language and are executed in the Dalvik virtual machine (DVM) that is substantially different from the classical Java virtual machine (JVM) [54]. The DVM is developed by Google and optimized for the characteristics of mobile operating systems (especially for the Android platform). The differences between Java and Dalvik bytecode based on JVM and DVM, respectively, are summarized in Table 3.3.

Table 3.2: Hardware and software specifications of experimental nodes

Platform	**Google NEXUS 7**
CPU	NVIDIA Tegra 3 quad-core processor (1.7 GHz single / 1.6 GHz quad)
Memory	1 GB, RAM
Storage	16 GB, Nand flash
Network	Wi-Fi 802.11 b/g/n, Bluetooth, NFC
Mobile OS	Android 4.2, Jelly Bean (Build number: JDQ39)
Kernel	Linux 3.1.10
Linux extension	Ubuntu 12.04 for ARM
JVM	JDK 1.6.0_32 (Oracle Java SE for Embedded 6u32 ARMv7 Linux)
Hadoop	1.1.2 stable release
Resource monitoring	Sysstat 10.0.3-1 stable version

Porting Hadoop framework on the Android operating system was a big and significant challenge at the early stage of this work. Since Android employs the Dalvik virtual machine to support its mobile applications, Hadoop software framework based on a specific Java virtual machine is not fully compatible with the Android runtime environment. Hence, Hadoop can be ported by either converting from its JVM based source codes and libraries to DVM compatible ones or installing a specific JVM recommended by the Hadoop project to run the original Hadoop software.

Most of the previous work [32, 33, 34] had difficulties with rewriting Hadoop codes for Android. They implemented only a small number of Hadoop functions by removing many core features that are incompatible with the Dalvik environment. In contrast to earlier approaches, this work successfully installed Oracle JVM that is a base platform for perfectly running Hadoop framework on Linux-based operating systems by adding a Linux extension [55], Ubuntu 12.04, to the Android Linux kernel as illustrated in Figure 3.2; this kind of approach is called Mobile virtualization, in which virtualization technology enables multiple operating systems or virtual machines to run simultaneously on a mobile device [56]. It was carefully ensured that there was no degradation of the hardware performance or adverse effect on Android operations. As a result, the experimental mobile cluster runs all the existing

Table 3.3: Comparison of Java and Dalvik bytecode

	Java bytecode executed in JVM	**Dalvik bytecode running in DVM**
Application Structure	Consists of one or more `.class` files, one file per class.	Has a signle `.dex` file containging all classes.
Register architecture	Stack-based. Push local varialbles onto a program stack for manuipulation.	Register-based. Assigns local variables to any of the 2^{16} available registers and directly manipulate registers.
Instruction set	Has 200 opcodes. Have a dozen of opcodes dedicated to moving elements between the stack and local variable tables.	Has 218 opcodes. Has longer instructions, since most of them contain source and destination address of registers.
Constant pool structure	Replicates elements in constant pools within the multiple `.class` files, e.g., referrer and referent method names.	Uses a single pool that all classes simultaneously reference and eliminates some constants by inlining their values directly into the bytecode.
Ambiguous primitive types	Variable assignments distinguish between integer (`int`) and single-precision floating-point (`float`) constants and between long integer (`long`) and double-precision floating point (`double`) constants.	Variable assignments (`int/float` and `long/double`) use the same opcodes for integers and floats, e.g., the opcodes are untyped beyond specifying precision.
Null references	Has a `null` type.	Does not specify a `null` type, instead opting to use a zero value constant.
Object references	Uses typed opcodes for the comparison of object references (`if_acmpeq` and `if_acmpne`) and for `null` comparison of object references (`ifnull` and `ifnonnull`).	Uses a more simplistic integer comparison between two integers, and a comparison of an integer and zero, respectively.
Storage of primitive types in arrays	Uses defined, unambiguous opcodes.	Uses ambiguous opcodes to store and retrieve elements in arrays of primitive types (e.g., `aget` for `int/float` and `aget-wide` for `long/double`).

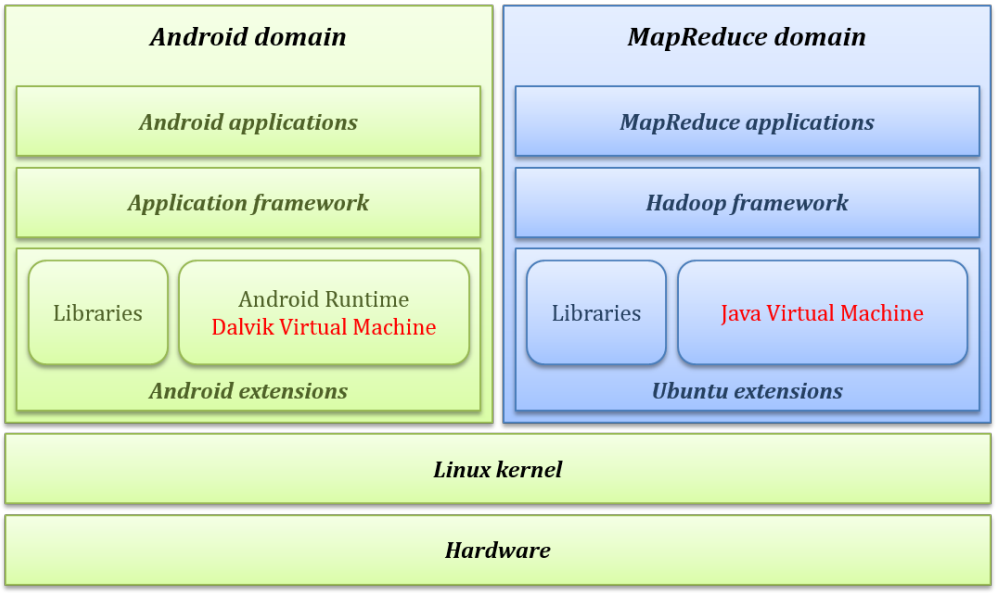


Figure 3.2: Mobile virtualization for MapReduce implementation

and emerging features of the Hadoop architecture, including MapReduce 2.0, also known as YARN [16].

The experimental mobile cluster that runs the Hadoop software is composed of a single master node and ten slave nodes which are configured with the default values for system parameters of the Android OS and Hadoop framework. The master node coordinates the slave nodes to get the workload done and the slaves run the sub-workloads, Map and Reduce tasks, assigned by the master node. The usage of computing and networking resources on each node is carefully monitored with a performance monitoring tool, Sysstat. To investigate node's behavior in the Hadoop workflow, two typical workloads – WordCount and TeraSort – are tested with associated Hadoop benchmark tools on the mobile cluster.

- WordCount: this workload counts the occurrence of each word in the input data sets generated by the Hadoop RandomTextWriter tool. It represents workload that extracts small but valuable analytics from the input data.

Table 3.4: Hadoop configurations of the experimental mobile cluster

Configuration metrics	Settings / values
Number of slave nodes	1 to 10
Available Map slots	2 per node
Available Reduce slots	2 per node
Workloads	WordCount, TeraSort (and TestDFSIO for I/O test)
Data block size	1, 2, 4, 8, 16, 32, 64 MB
Input data size	1 to 10 GB
DFS replication	1

Note: MapReduce slots define the maximum number of Map and Reduce tasks that can run in parallel on a cluster node. The number of slots on each cluster node may be different and it depends on the node's hardware capacity.

- TeraSort: this workload sorts the input data sets generated by the Hadoop TeraGen tool in a predefined order. It represents workload that transforms unstructured source data to a more useful structure or format without adding new data.

The input and output data usually need to be replicated to a small number of physically separate nodes (typically three) to insure against data block corruption and hardware failure. However, this work disables the replication of both input and output data in the experiments to concentrate on core behaviors of the MapReduce workflow. The details of the configuration metrics are listed in Table 3.4.

3.3.2 I/O performance of mobile nodes

Before analyzing the performance of Hadoop framework on the mobile cloud cluster, the experimental cluster nodes are tested to examine their robustness, availability, and error handling under a heavy load and to investigate which resource (e.g., CPU, memory, filesystem, network, etc.) may affect the performance of MapReduce applications. In the stress testing, remarkable performance characteristics of the filesystem and network I/O are observed, in which the Hadoop TestDFSIO benchmark that tests the I/O performance of the distributed filesystem by sampling the number of bytes read/written at fixed time intervals is utilized

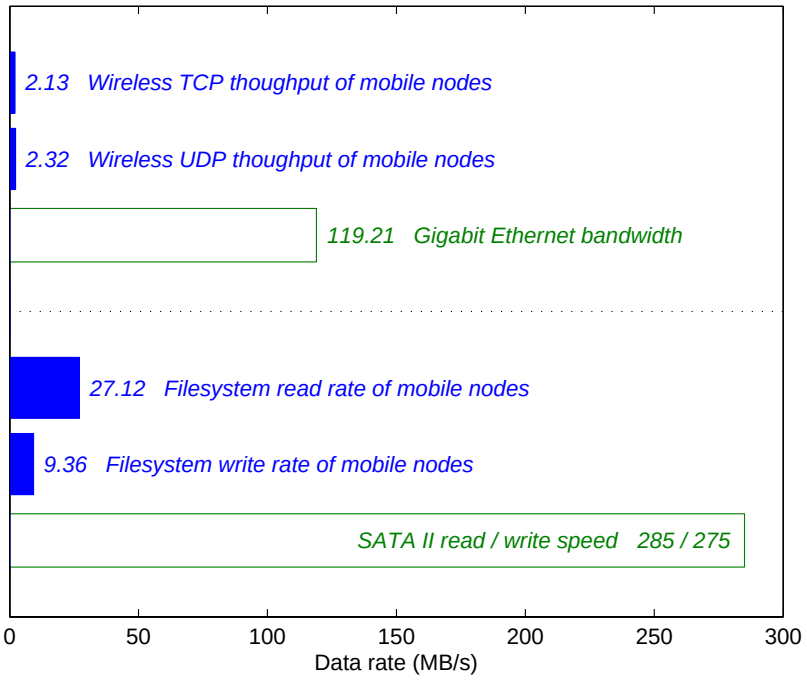


Figure 3.3: Network and filesystem throughput of mobile nodes

to measure the actual HDFS read/write speed (in a single-node cluster setup) and the Iperf network performance measurement tool that generates constant TCP or UDP traffic flows is employed to compute the actual network throughput (between two cluster nodes).

Figure 3.3 displays throughput measurements of the filesystem and network in the load tests. The result shows that the network and filesystem I/O of mobile devices are far slower than those of commodity servers with one or two wired Gigabit connections and SATA II internal storages; 10-Gigabit Ethernet and SATA III storage are already common. In particular, the available network throughput of mobile nodes is too much lower than that of typical Hadoop clusters presented in [18]. The network speed is mush slower than the data transfer rates of internal storage as well. Since the actual effect of the network bandwidth on Hadoop performance is relatively small in conventional Hadoop setups with high-speed wired connectivity, not much attention has been paid to Hadoop operations under the network bandwidth constraint that is critical for reliable data transfers.

Consequently, the performance of mobile cloud clusters may be strongly influenced by the characteristics of wireless links in the operating environments. Although computing capabilities of cluster nodes are a significant performance factor, each node also needs the capability to read and write large amounts of data to and from the distributed filesystem that is implemented on remote nodes. In wireless networks with relatively low network bandwidth, time required to transfer data blocks can significantly contribute to the total processing time even though the distributed computing power generally decreases the amount of time required for job completion.

3.3.3 Performance of WordCount workload

The WordCount workload that counts the occurrence of each word in the input data sets produces small final output. The Map phase is generally computation intensive, compared to other phases. Network utilization is low in the Shuffle phase, in which the Map tasks transfer their output (i.e., intermediate results) to the Reduce task as input, because the Map output is a small subset of the large input data set in this kind of workload.

Figure 3.4 shows the network utilization with MapReduce task progress of the WordCount workload that starts with 1 GB input data. In the workload, 20 Map tasks corresponding to the 1 GB input size are equally distributed over 10 slave nodes. One node is chosen to run the single Reduce task that produces the final output. Figure 3.5 displays resource utilization on two typical slave nodes; the Map node runs only two of 20 Map tasks and the Reduce node runs both the Map tasks and the additional Reduce task.

Figure 3.4 contains an aggregate data traffic pattern receiving from all nodes running Map tasks, which is denoted by the solid line and a single data flow transmitted by a typical Map node, denoted by the dash line. The graph shows two bursts of received traffic since each node finishes two assigned Map tasks one at a time and transmits the intermediate result at the same time to the single node running the Reduce task.

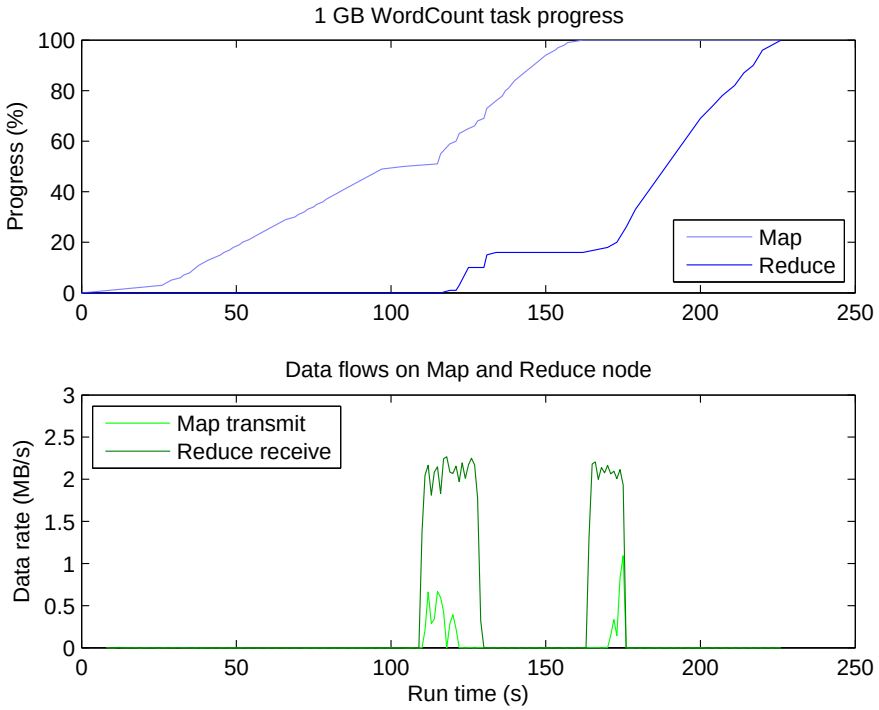
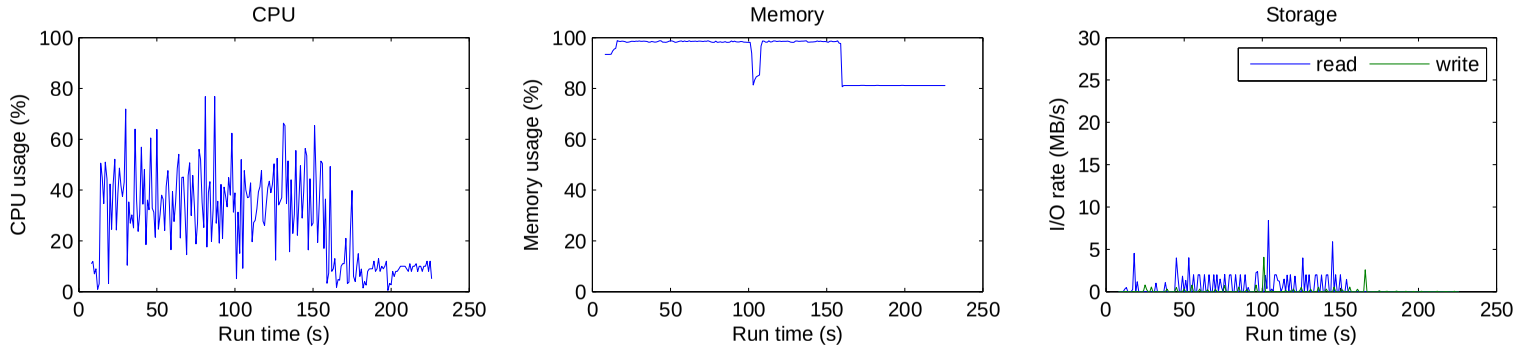


Figure 3.4: Network utilization of Hadoop mobile cluster with WordCount workload

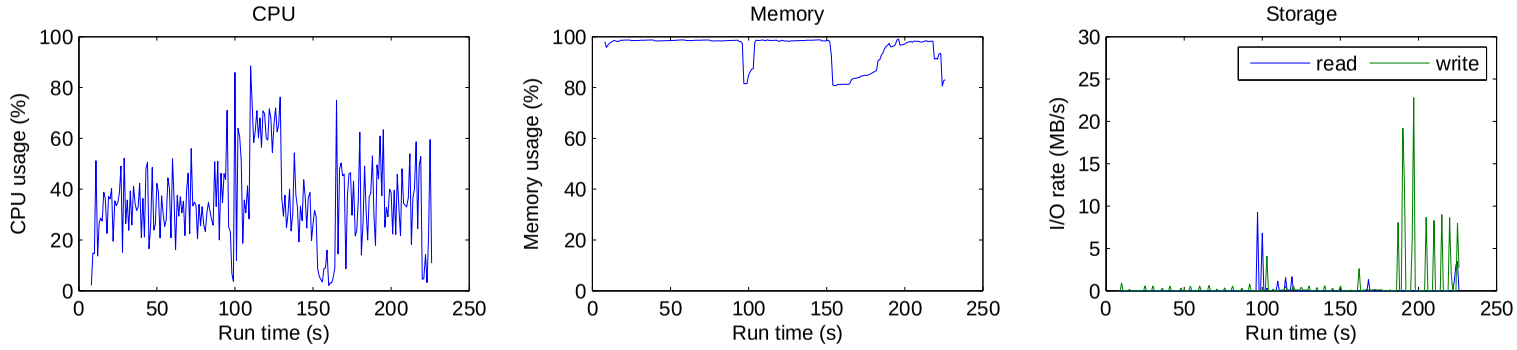
Although Hadoop has the ability to process multiple tasks simultaneously within resource bounds, the experimental nodes run tasks sequentially due to lack of memory (see Figure 3.5). This explains the separated bursts of traffic and corresponding delays in the Map and Reduce progress. The network bandwidth is saturated during each burst, but it only lasts for a short period of time since the output of the Map tasks is very small.

3.3.4 Performance of TeraSort workload

The TeraSort workload that sorts input data sets generates a large amount of intermediate data in the Map phase, which needs to be transmitted to the Reduce task over the network to produce the final output. Both Map and Reduce phase are commonly computation and I/O intensive. Network utilization is very high in the Shuffle phase because the output of Map tasks has the same size as the input data sets in this workload.



(a) Map node



(b) Reduce node

Figure 3.5: Resource utilization of MapReduce nodes running WordCount tasks

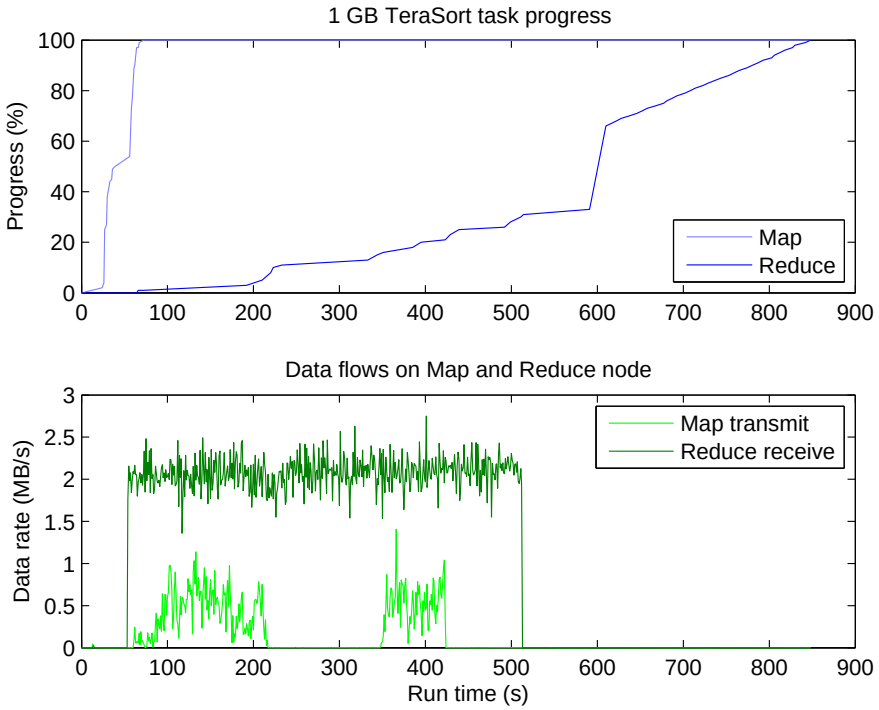
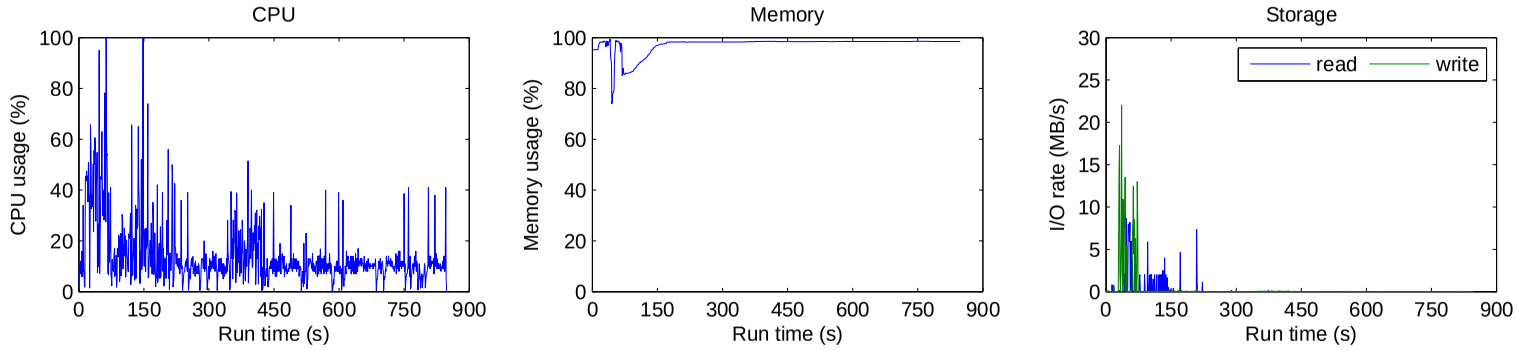


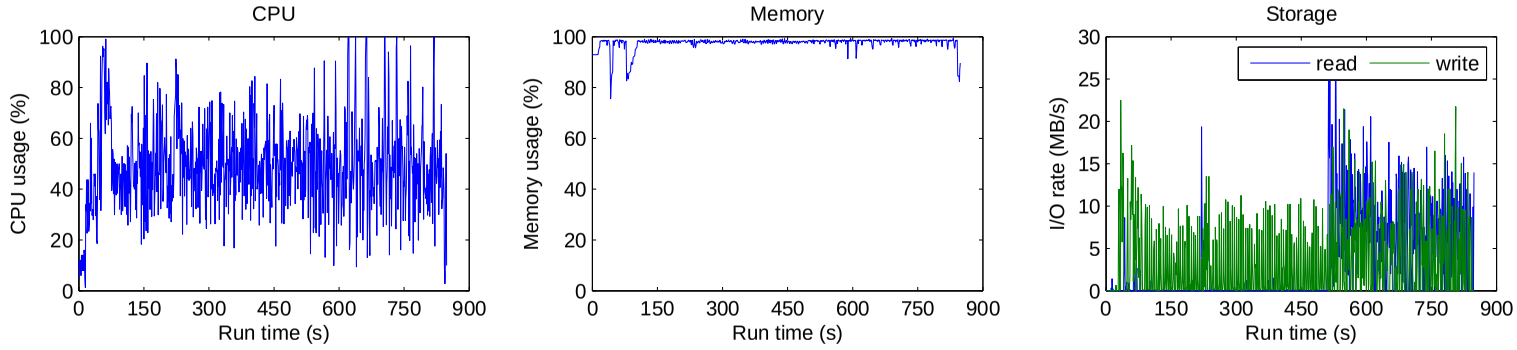
Figure 3.6: Network utilization of Hadoop mobile cluster with TeraSort workload

Figure 3.6 shows the network utilization with MapReduce task progress of the TeraSort workload initialized with 1 GB input data. The configuration is identical to the WordCount workload; 20 Map tasks are equally distributed over 10 slave nodes and one node runs the single Reduce task. The resource utilization of two different slave nodes is detailed in Figure 3.7 in the same way as the WordCount performance analysis.

Figure 3.6 illustrates a large volume of aggregate traffic made up of data flows transmitted at the same time by multiple nodes because the entire input data needs to be shuffled to the single node running the Reduce task. The network bandwidth is saturated while the output of all Map tasks is being transferred. This traffic pattern increases the possibility of packet loss, resulting in throughput reduction and fluctuating performance; a significant number of TCP packets are dropped during the shuffle phase. Consequently, the Map tasks finish relatively quickly but the Reduce task makes slow progress since it spends a great deal



(a) Map node



(b) Reduce node

Figure 3.7: Resource utilization of MapReduce nodes running TeraSort tasks

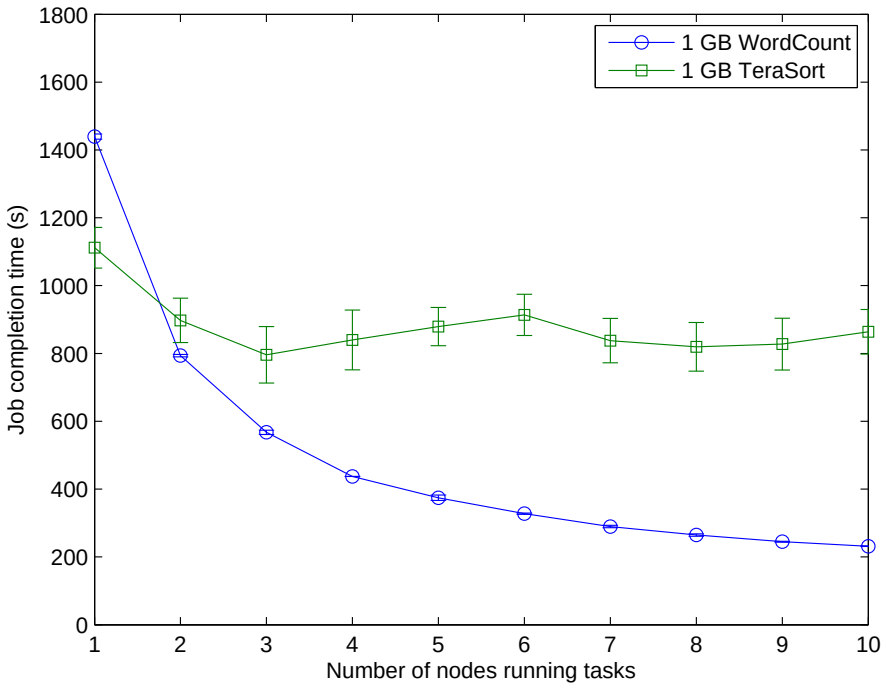


Figure 3.8: Cluster size scaling of WordCount and TeraSort experiments

of time in receiving the large input data (i.e., the output of Map tasks) and processing the entire data sets.

3.3.5 Performance of scale testing

This section examines the effects of scaling up the cluster size, data block size, and input data size that represents the variability in configuring the mobile ad hoc cloud.

First, an optimally configured cluster generally has the ability to improve performance by scaling up the cluster size. Figure 3.8 shows the results from the experiments which are intended to verify how the cluster size affects performance of the mobile distributed framework. The job completion time of two typical workloads, WordCount and TeraSort, with 1 GB input data is measured as the number of slave nodes participating in the cluster gradually increases.

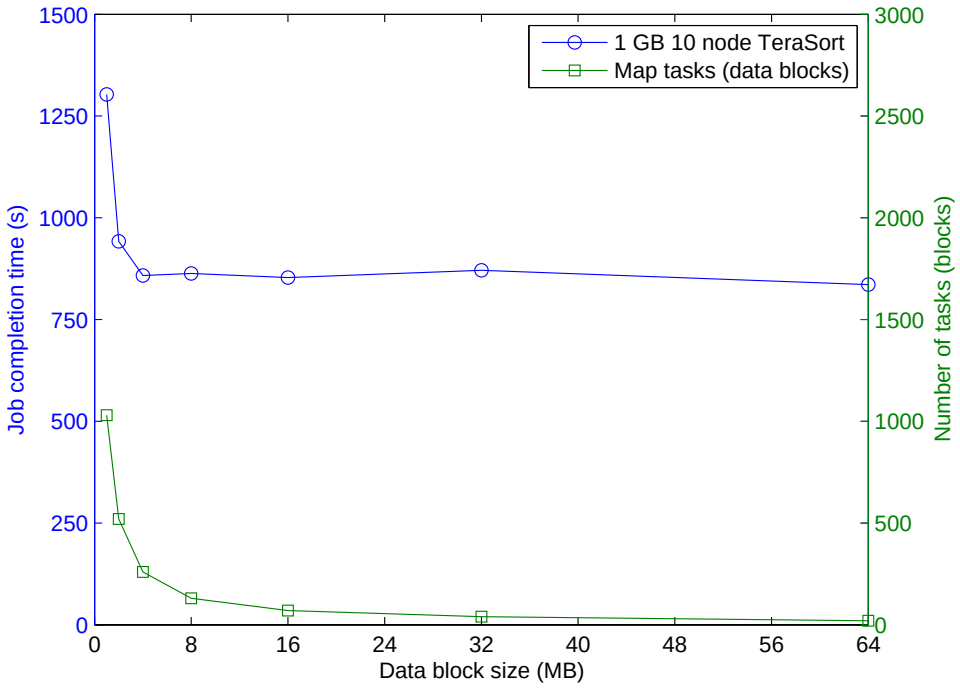


Figure 3.9: Data block size scaling of TeraSort experiments

As indicated in Figure 3.8, increasing the number of nodes considerably decreases the job completion time of the WordCount workload. On the other hand, in the cluster scaling with the TeraSort workload, the increase in cluster size does not lead to a significant decrease in job completion time because the performance of the mobile cluster is bounded by the time taken by the entire input data to be shuffled under the limited network bandwidth that is also highly variable.

Second, the unit of input for a Map task is a data block of the input file. A single large input file is split into many blocks which are distributed over the nodes in the Hadoop cluster. The size of a data block stored in Hadoop filesystem is large – 64 MB by default, compared to a block size in traditional filesystems – normally 512 bytes. By making a block large enough, the data transfer time from the disk becomes significantly larger compared to the time required to seek the start of the block. Thus, the transfer operation of a large file made of multiple blocks becomes faster by minimizing the seek time [16].

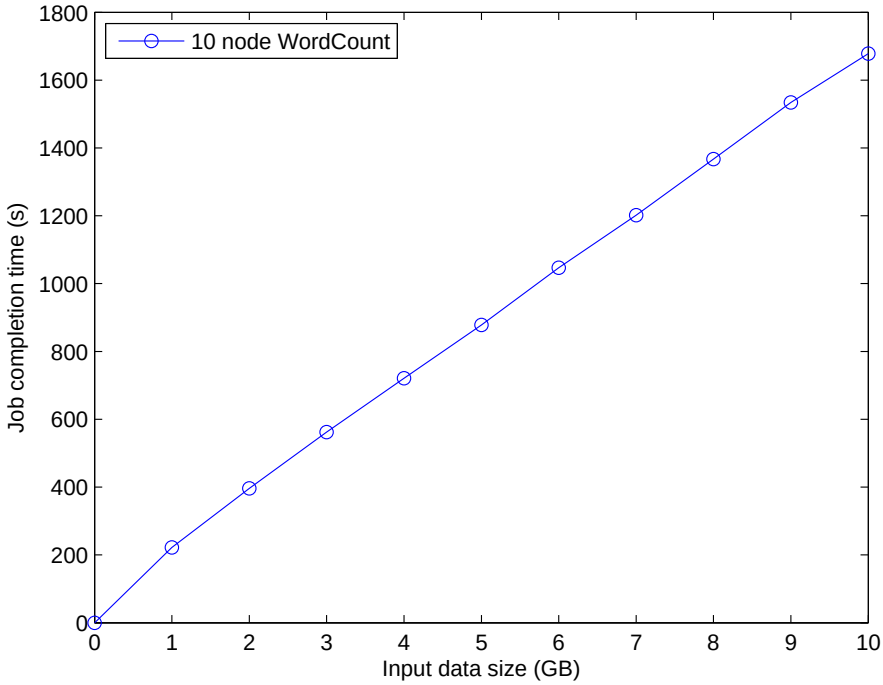


Figure 3.10: Input size scaling of WordCount experiments

What is the effect of the data block size in wireless configurations where one or more phases of MapReduce transfer a considerable number of data blocks over wireless links with low throughput? The previous study, Hyrax [32], suggested the use of a small block size in consideration of the lengthy transfer time and delay of the large block in the wireless network. However, they did not provide any comparative measurements to validate their suggested value. To determine an appropriate data block size for the Hadoop mobile cluster, the job completion time of the I/O intensive TeraSort workload with 1 GB input data is measured as the data block size gradually increases.

Contrary to expectations, Figure 3.9 displays performance degradation in small data block sizes. A Map task handles a data block of input at a time. If the data block is very small (i.e., there are a large number of data blocks), more Map tasks are required to process each data block as also shown in the figure. This imposes an inefficient data access pattern by causing frequent seeks to retrieve each small block. Furthermore, resources are scarce for

an excessive number of Map tasks. Hence, configuration parameters for the mobile cluster should be carefully determined by taking into account various other performance aspects.

Finally, Figure 3.10 demonstrates the impact of input data size on the job completion time of the WordCount workload as the size of the input data increases. The larger the input data, the longer it takes to process the workload and produce the output result. Meanwhile, a problem has been encountered when plotting the same measurements from the TeraSort workload because its performance is extremely variable and unreliable due to an increasing number of task failures (caused by task response timeouts and intermittent node disconnections) and re-runs. This work identifies the cause of the failures in the following chapters.

3.4 Performance simulations of Hadoop mobile clusters

Comprehensively evaluating the performance in extensive operating setups is important to understand the performance issues of mobile cloud clusters and the overall efficiency of the distributed system. However, it is increasingly harder to evaluate and repeat every possible configuration as the scale of the cluster size become larger before gaining insight into the system performance. For example, to make cluster nodes move coordinating a large number of nodes and to observe performance constrains in various wireless communication conditions are necessary, but performing the actual experiments involves both time and cost. For a similar reason, many researchers have proposed simulation-based solutions to evaluate and optimize MapReduce systems by saving cost and time. This work also shares the concern of a need for simulation approaches and develops a MapReduce simulator for mobile cloud clusters based on one of existing simulators, MRPerf. Using the simulator, extensive MapReduce simulations are conducted to identify more performance issues under different MapReduce scale, node mobility, and wireless channel condition.

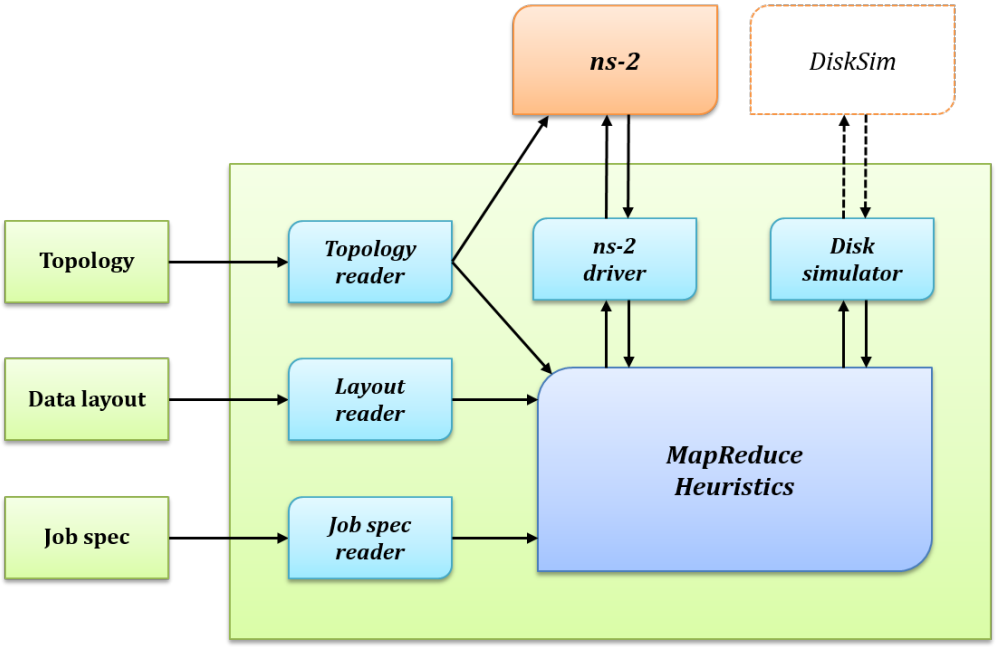


Figure 3.11: Architecture of MRperf simulator

3.4.1 MRPerf simulator for MapReduce

MRPerf [49] is one of the earlier simulator tools for the MapReduce data processing framework, which provides a fine-grained simulation to capture various performance aspects of MapReduce execution phases and predict application performance. Since MRPerf is based on the popular ns-2 network simulator [50], It models inter- and intra-rack task communications over the networks to simulate correct network behavior. Some of the important motivations behind MRPerf were to examine the performance of MapReduce applications under varying cluster configurations, different network topologies, different data placement algorithms and different task schedulers by assuming that a node's resources are equally shared among tasks assigned concurrently to the node and the simulator does not model OS-level asynchronous prefetching.

Figure 3.11 shows the high-level architecture of MRPerf. The input configurations for initializing the simulator are defined in a set of files, and processed by different processing modules. The ns-2 driver module provides the interface for network simulation. Similarly,

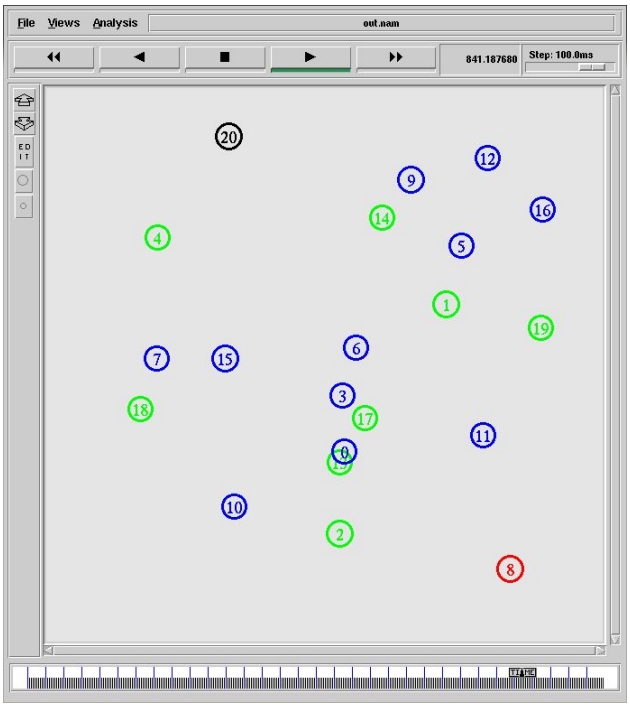


Figure 3.12: Screenshot of MapReduce simulation

the simple disk module that models the storage I/O can be extended to interface with other disk simulators. All the modules are driven by the MapReduce Heuristics module (MRH) that simulates Hadoop's behavior. To perform a simulation, MRPerf first reads all the configuration parameters and instantiates the required number of simulated nodes arranged in the specified topology. The MRH then schedules tasks to the nodes based on the specified scheduling algorithm. This enables each node to run its assigned tasks, which further creates network traffic (modeled through ns-2) as nodes interact with each other. Thus, a simulated MapReduce workload is created.

3.4.2 Implementation and validation of MapReduce simulator

This work selects MRPerf as a base simulator to implement MapReduce simulation for mobile cloud clusters since it is open source and supports various cluster configurations and network topologies by utilizing the network simulator. MRPerf was originally developed to investigate the impact of network topologies on the performance of Hadoop clusters that

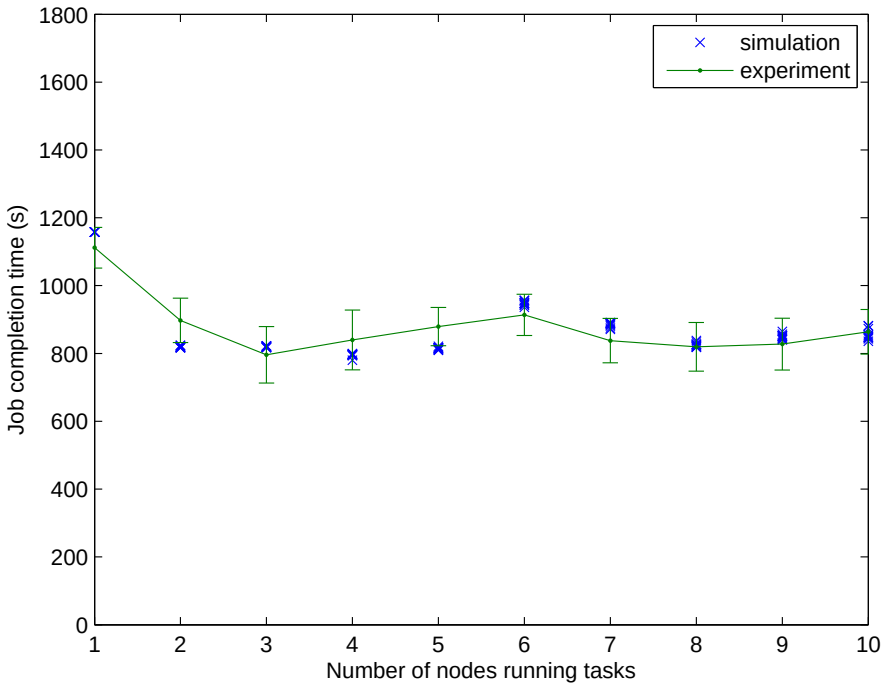


Figure 3.13: Validation of MapReduce simulation with TeraSort workload

consist of commodity servers. All the network topologies (e.g., Star, Double rack, Tree and DCell) represent switch-based wired network setups and all the cluster configurations are based on node characteristics (e.g., CPU, memory, disk, etc.) from server platforms. Thus, the mobile cluster's characteristics and behaviors under wireless ad hoc configurations need to be integrated into the MRPerf framework.

Implementing a mobile cluster on MRPerf involves a lot of tasks. First, a careful analysis of MRPerf architecture and source codes was conducted. Second, cluster configuration parameters for the mobile clusters, such as hardware capacity and Hadoop/MapReduce setup, were attentively tuned. Finally, additional features and exception handlings for node mobility were implemented. Figure3.12 displays a sample screenshot of the MapReduce simulator for mobile cloud clusters, where 20 mobile nodes are moving in a simulation area by running a MapReduce application; black circle denotes a master node, blue and green circle denotes a slave node with and without running tasks, and red circle denotes a malfunctioning node.

Table 3.5: Simulation configurations of the Hadoop mobile cluster

Configuration metrics	Settings / values
Simulation area (meter×meter)	50×50, 100×100, 200×200, 300×300, 400×400, 500×500
Radio propagation model	FreeSpace, Shadowing, TwoRayGround
Node mobility	No mobility (static nodes), Random Waypoint
Number of slave nodes	1 to 40
Available Map slots	2 per node
Available Reduce slots	2 per node
Workloads	TeraSort
Data block size	64 MB
Input data size	1 to 10 GB
DFS replication	1

For the validation tests, this work compares data collected from real cluster configurations running 1 GB TeraSort workload with data observed in the MapReduce simulation runs with the same workload. Each cluster node has wireless single-hop connections and is stationary during the tests, which allows the validation tests to consistently evaluate the performance of the cluster under reliable connectivity. The job completion time is measured as the number of cluster nodes gradually increases. Figure 3.13 shows the results from the actual experiments as well as simulation runs. As indicated in the figure, the MapReduce simulator is able to predict the job completion time within the margin of error. Through a series of validation tests, this work finds that it is capable of simulating the MapReduce performance of mobile clusters fairly accurately.

3.4.3 Performance of scale testing

Using the MapReduce simulation, this section investigates the effects of scaling up the cluster size, input data size, and simulation area size to provide more insights into the mobile ad hoc cloud. The details of the configuration metrics are listed in Table 3.5.

First, Figure 3.14 displays the result of simulations designed to verify how the cluster size affects the performance of I/O intensive MapReduce applications. The job completion

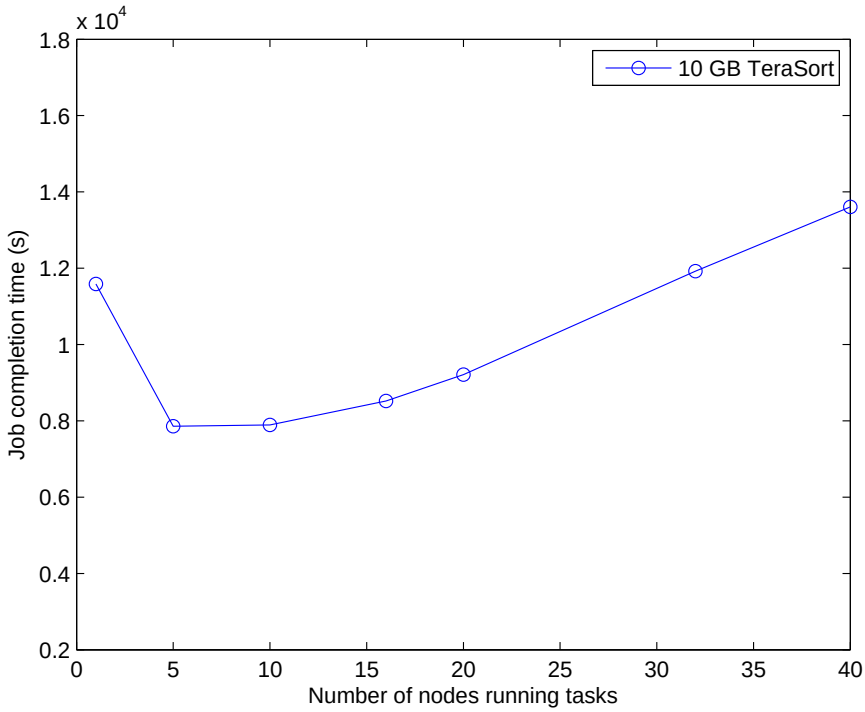


Figure 3.14: Cluster size scaling of TeraSort simulations

time of TeraSort workload with 10 GB input data is measured as the number of cluster nodes increases up to 40. As indicated in figure, increasing the number of nodes decreases the job completion time of the TeraSort workload but it does not lead to a significant decrease in job completion time since the performance is bounded by the time taken by the large amount of intermediate data to be shuffled under the wireless communication links as discussed in the earlier section.

Second, Figure 3.15 demonstrates the impact of input data size on the job completion time of TeraSort workload in the 10 node cluster setup as the size of the input data increases up to 10 GB. The result exactly shows that the execution time is proportional to the size of input data. Note that the actual performance of TeraSort workload in large-scale input data setups was unmeasurable because unexpected problems of system operation were frequently encountered before getting a final result and the performance measures were highly variable during the experiments using real mobile devices. Thus, the MapReduce simulator can be

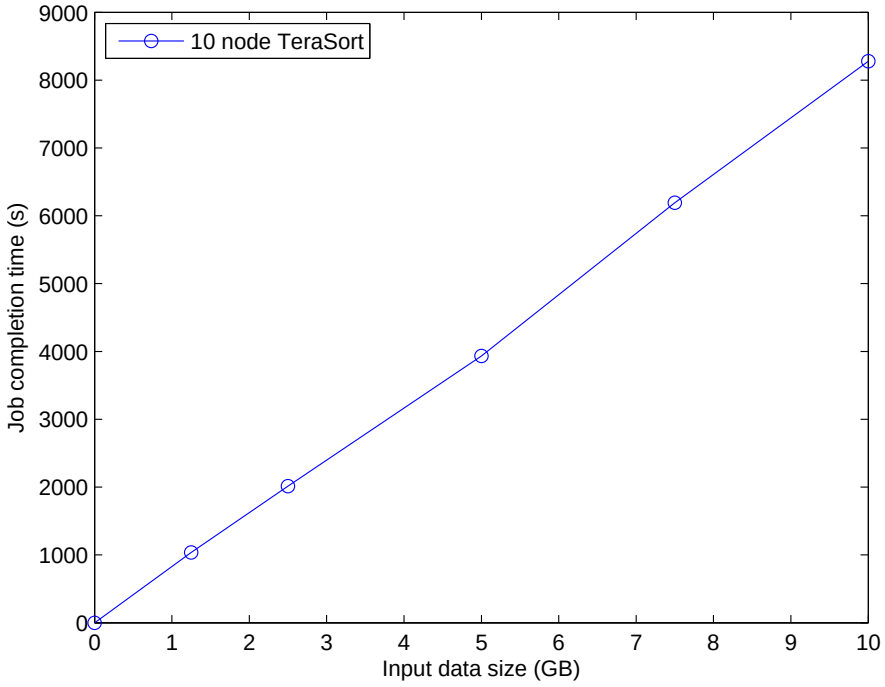


Figure 3.15: Input size scaling of TeraSort simulations

a useful tool to predict the performance trend by avoiding temporarily unobservable faults from other performance factors that are not relevant to the MapReduce execution.

Lastly, Figure 3.16 shows the effect of the size of simulation area. In the simulation, the mobile cluster with 10 nodes runs 1 GB TeraSort workload. The effective communication range of wireless cluster nodes is set to around 100 meters in the simulation considering the actual radio coverage of mobile devices. As a result, cluster nodes are frequently disconnected in a large simulation space due to range-limited communication, which means that the Hadoop cluster experiences numerous task timeout events as timely task monitoring is unavailable, resulting in job execution failure. Interestingly, the simulation result shows that it is possible to run MapReduce applications with node mobility by adjusting Hadoop configuration parameters (e.g., the heartbeat interval and task timeout period), which enables the intermittently connected mobile nodes to perform cooperative processing in the Hadoop framework.

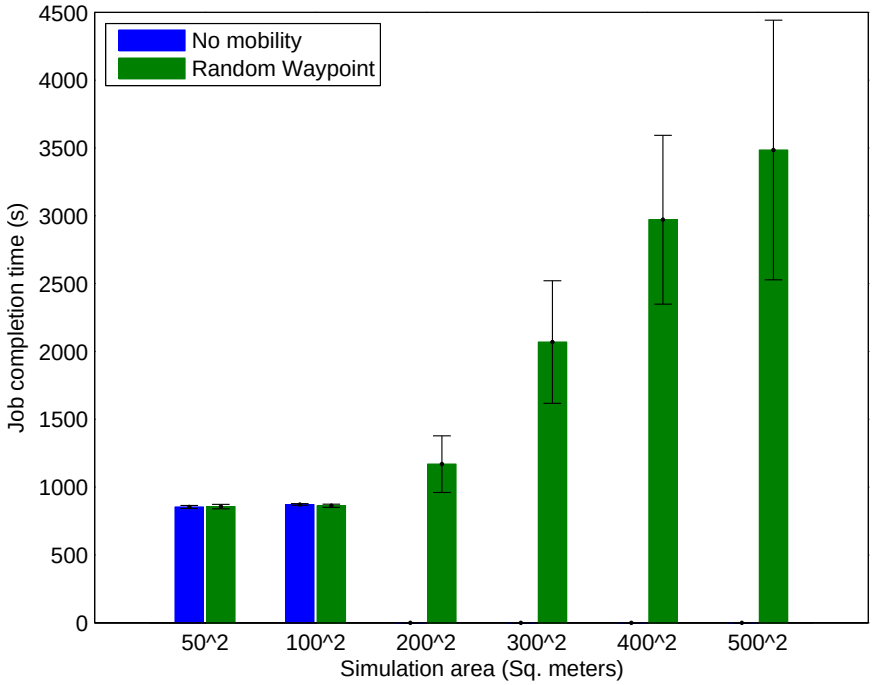


Figure 3.16: Simulation area scaling with node mobility

3.4.4 Performance over different radio propagations

Wireless network simulation is an important tool for evaluating wireless protocols and wireless applications before they evolves towards real world implementation. Many studies have used network simulators, such as ns-2, for the simulation of wireless networks. The ns-2 simulator supports different Physical (e.g., radio propagation) and MAC layer models (e.g., IEEE 802.11 standard), network routing protocols (e.g., AODV, DSR, DSDV and OLSR), and their extensions (or modifications). Since the MapReduce simulator exploits the ns-2 for simulating application's performance over different network configurations, it is possible to apply various wireless network models and protocols for the MapReduce simulation.

The wireless networks are much more complicated to analyze than wired networks. Their characteristics and conditions may change rapidly and randomly. There are significant differences between simple wireless paths with line of sight (LOS) and those with obstacles (e.g., vehicles and buildings) between the sender and receiver as well. To investigate the

impact of various wireless channel conditions, this work compares the performance of the Hadoop mobile cluster with different propagation models.

The network simulations generally consider two channel models: large-scale and small-scale propagation models. The large scale propagation models predict the mean signal strength for an arbitrary transmitter-receiver separation distance to estimate the radio coverage area of the transmitter. On the other hand, the small scale models characterize the rapid fluctuations of the received signal strength over very short travel distances or short time durations. Due to multipath propagation of radio waves, movements of the receiver can have large effects on the received signal strength.

This work tests three frequently used radio propagation models included in the ns-2 network simulator for the mobile cluster simulation. First, FreeSpace model assumes that the received power is only dependent on the transmitted power, the antenna's gains and on the distance between the transmitter and the receiver, where a radio wave that moves away from the sender has to cover a larger area so the received power decreases with the square of the distance. Second, Shadowing model assumes that the average received signal power decreases logarithmically with distance, where a Gaussian random variable is added to this path loss to account for environmental influences at the transmitter and the receiver. Third, TwoRayGround model assumes that the received energy is the sum of the direct line of sight path and the path including one reflection on the ground between the sender and the receiver.

Figure 3.17 demonstrates the result of simulation runs (where 10 cluster nodes runs 1 GB TeraSort workload in two simulation area sizes) with different propagation models, which indicates that the radio wave propagation has a strong impact on the job completion time. Actual mobile devices are equipped with heterogeneous wireless adapters and antennas, which means that mobile nodes have different transmit and receive capabilities and their mobility induces complicated changes to the communication coverage. Thus, the received signal strength of mobile devices may vary over the time periods of communication (i.e.,

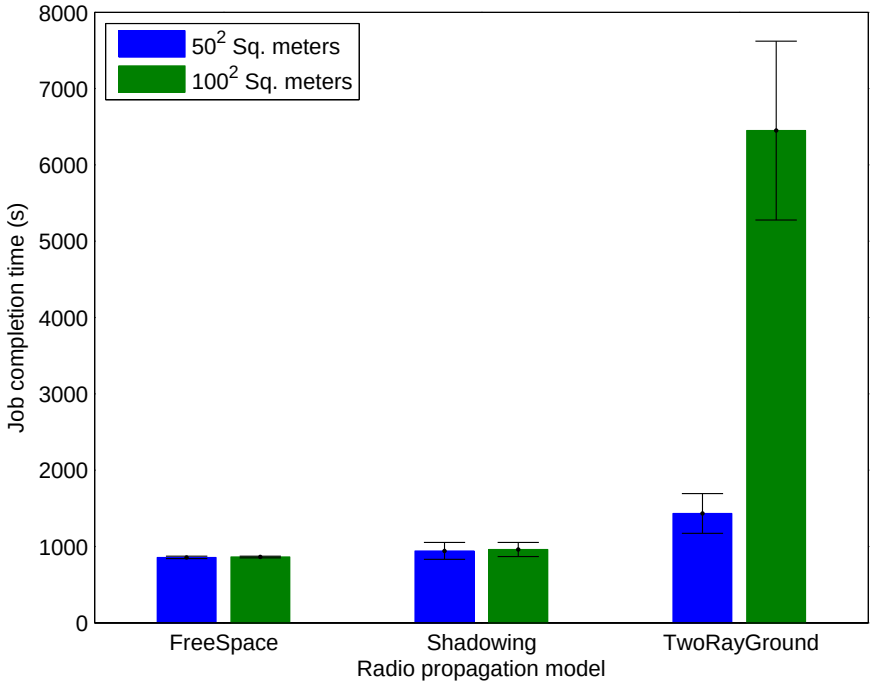


Figure 3.17: MapReduce simulations with different radio propagation models

data transfers). The Shadowing model simulates this case and the simulation result shows that variation on the job completion time increases in comparison to one with the FreeSpace model. Furthermore, it is also observed that the multipath propagation of radio waves simulated by the TwoRayGround model has a significant effect on the performance of the mobile cluster, especially with a larger simulation space (i.e., an increased communication range), since the propagation pattern causes serious communication problems such as jitter and ghosting.

3.5 Performance issues of Hadoop mobile clusters

In the performance analysis it is found that the overall computing power of the mobile cluster is no longer significantly bounded by internal resource capabilities of each individual node since mobile devices have been constantly enhancing their resources and processing power. On the other hand, this work identifies distinct performance problems in MapReduce

analytics on the Hadoop mobile clusters, which come in the form of longer job completion time or frequent task failure from task response timeout and node disconnection.

In distributed systems where a controller usually makes control decisions with limited information from remote components, a timeout control provides a key mechanism through which the controller can infer valuable information about unobservable states and events in the system when direct feedback is either impossible or costly [57]. The timeout control is configured using a timer which expires after a timeout threshold. This defines an expected time by which a specific event should occur. If no information arrives within this period, a timeout event occurs and the controller triggers corresponding reactions. In fact, the timeout control is an integral component for building up a reliable distributed system.

The Hadoop distributed system also adopts the timeout control for both job scheduling and progress monitoring. A MapReduce job initiates long-lived batch tasks running on slave nodes, which usually take a few minutes or hours. Because of the significant length of run-time, it is important for the master node to get feedback on how the job is progressing in a timely fashion. It enables the master to keep track of task status and restart failed or slow tasks. If a slave (task) fails by crashing or running very slowly, for example, it stops sending (or sends intermittently) current status and progress updates, called heartbeats, to the master; the master then marks the slave (task) as failed after the timeout threshold which is 10 minutes by default [16].

In the previous experiments and simulations, the frequent timeout occurrences (task failures) with corresponding performance degradation while running the I/O intensive Tera-Sort workload with large input data were observed in the Hadoop mobile clusters. The problems can be summarised as follows:

First, the job execution time is sensitive to slow-running tasks as only one slow task makes the time significantly longer. When a mobile node running Map tasks has significant delays in transmitting a large amount of intermediate result to Reduce tasks through wireless connections (i.e., tasks are running slower than expected due to the lengthy transfer time

of Shuffle phase), the master launches another, equivalent tasks as a backup instead of diagnosing and fixing the slow-running tasks. The slow-running (or hanging) tasks are considered failed and automatically killed after the timeout period. The master also tries to avoid rescheduling the tasks on the slave node where they have previously failed.

Second, depending on the size of the cluster, the master node has high resource requirements as it manages the cluster resources, schedules all user jobs, and holds block metadata of the distributed filesystem. On a busy cluster running a heavy workload, the master uses more memory and CPU resources. Thus, the master node based on a mobile device is subject to resource scarcity and bottlenecks in processing received data in a timely fashion. When the master has not received an expected progress update from a slave node for the timeout threshold, it arranges for all the Map tasks that were scheduled on the failed node, whether completed or not, to be rerun since intermediate output residing on the node may not be accessible to the Reduce task.

Consequently, theses failures and reactions lead to a significant increase in job execution time. Therefore, it is critical to mitigate the effect of the timeout occurrences in the Hadoop mobile clusters where the chance of particular node failures and communication problems is comparatively high.

Chapter 4

Problem statements and research questions

Most of the current distributed systems including Hadoop employ Transmission Control Protocol (TCP) for reliable communications between cluster nodes. The performance of mobile distributed processing largely relies on how effectively each mobile device exploits the available network resources through TCP connections as outlined earlier. Despite advances in mobile technologies, mobile devices still face significant limitations on transmitting and receiving reliable TCP data streams required to avoid any interruptions while performing distributed analytics.

4.1 Limitations on TCP performance over mobile devices

Mobile devices use a wireless channel as a transmission medium. Unlike wired networks, the time-varying condition on the wireless channel is the dominant cause of packet loss. TCP proposals mostly designed for wired networks are unable to react adequately to the packet loss due to channel noise, fading, or interference since they assume the only source of packet loss is congestion [20]. The random packet loss in the wireless channel makes it difficult for mobile nodes using one of those proposals (e.g., TCP CUBIC [25] in Android OS based on Linux Kernel) to estimate available channel bandwidth and achieve optimal TCP throughput. In addition, most of the wireless protocols allow wireless devices to share the same channel through contention-based media access control (MAC) that includes processes for initiating a new transmission, determining the channel state (e.g., available or unavailable), and managing retransmissions in the event of a busy channel or data loss, which has several limitations. If many nodes attempt to transmit data at the same time, for example, a substantial number of collisions may occur and result in lowering the available

bandwidth. Without pre-coordination, it is hard to prioritize data flows and prevent unfair transmissions. Not many studies have been made on TCP performance of mobile distributed analytic applications under these practical constraints.

The IEEE 802.11 standard for WLANs [58] defines several Physical-layer (PHY) data rates (e.g., most of recent mobile devices supporting IEEE 802.11n [53] use eight data rates: 6.5, 13, 19.5, 26, 39, 52, 58.5, and 65 Mbps) to provide more robust communication by falling back to a lower rate in the presence of a high noise level, where a rate adaptation algorithm of Media access control (MAC) layer makes a runtime prediction of changes in the channel condition and a selection of the most appropriate PHY rate. Although the PHY rate change is critical to the TCP performance, the cross layer interaction between the TCP flow control and MAC rate adaptation is yet to be thoroughly investigated [59]. For example, a problematic issue arises when the rate adaptation algorithm aggressively and rapidly reduces the PHY rate due to short-term degradation of channel quality (e.g., 65 $\rightarrow$ 52 Mbps: 20% decrease, 52 $\rightarrow$ 26 Mbps: 50%). TCP reacts to the sudden PHY rate reduction but needs a substantial amount of settling time to converge into a stable rate by updating its congestion window size corresponding to the PHY rate after detecting packet losses. In the event that frequent rate changes occur in the PHY layer, it is hard to utilize the available bandwidth to the fullest extent using TCP. Moreover, the TCP performance may drastically deteriorate if inappropriate PHY rates are selected unnecessarily.

4.2 Problems of using mobile devices for mobile cloud

Some low-cost smartphones and tablets continue to have resource limitations compared to traditional PCs and laptops in spite of the advances made in their hardware capabilities. Their wireless capability, in particular, is limited by several factors including power-saving operations (which may result in lower communication quality and intermittent connectivity), form factor constraints (that involve challenges in antenna implementation and placement), and minimal production costs (that bring about small network buffer/queue sizes due to low

memory capacity), all of which subject them to throughput reduction and fluctuating performance in wireless communications [60]. Moreover, when an application on the receiver is not able to process incoming TCP packets as fast as senders transmit due to lack of processing resources, the receiver sets the TCP flow limit by reporting the decreased receive window size (e.g, no buffer space available in the worst case). As a result, the sender's transmission will eventually be bounded by the receiver's processing rate. Thus, the processing capability of a mobile device potentially becomes a significant performance factor (i.e., TCP transmit rate bound) in TCP data communications when the mobile device experiences resource scarcity on processing data.

Furthermore, most of the mobile devices are generally optimized to improve receive performance, which can be found when looking into the mobile OS kernel and network drivers. For example, the mobile devices have an asymmetric resource scheduling (or distribution) scheme for transmitting and receiving data, where the mobile kernel allocates more resources to speed up processing of data frames on arrival and only the minimum number of frames necessary to acknowledge the received frames is scheduled for transmission while receiving data. In addition, the mobile OS does not alert the user to runtime errors of its wireless kernel or hardware faults nor display information about the internal problems directly, which makes it difficult to identify critical performance problems and improve the performance of mobile applications. Besides, it is not an easy task to customize the OS kernel and wireless driver of mobile devices for the variable operating environments although the mobile OS is open-source. Hence, the network (or application) performance observed on mobile devices may not be optimal and it is hard to find out the performance bound.

4.3 Research questions

This work studies the advantages and challenges of utilizing mobile devices for distributed analytics by showing its feasibility and conducting performance analysis. The empirical study based on experiments and simulations focuses on how to build the mobile ad

hoc cloud by clustering with nearby mobile devices to reliably support practical distributed analytics. The following questions are addressed in this work:

- Is a typical distributed software framework for cloud computing capable of effectively supporting mobile distributed analytics? What are the problems of implementing the distributed analytic framework on mobile devices? How can the performance of distributed analytics be evaluated on practical mobile clusters?

- What are the limitations in enabling mobile devices to offload portions of the workload to remote computing resources and share their resources for distributed processing? How efficiently can the controller node initiate distributed analytics using dynamic mobile cloud resources under the time-varying operating environment?

- In what ways, is the mobile cluster able to mitigate the effect of frequent task failures while supporting large complex computations and long-running processes for distributed analytics, which are usually caused by hardware/software faults (or slow-running tasks) and communication problems?

- How can reliable data communications between mobile devices for analytical data transfers in the workflow of distributed analytics be guaranteed under the limitations of TCP performance over wireless links? What is the best way to control TCP flows on mobile devices for improving performance of mobile distributed analytics?

To resolve these questions, this work has developed a test bed of mobile distributed platforms and a simulator for mobile cloud clusters by conducting performance analysis to identify performance issues as presented in the previous chapter, and will propose adaptive TCP flow control algorithms for enhanced analytic performance by validating the proposed solutions through extensive experiments in the following chapters.

Chapter 5

Adaptive TCP flow control for mobile clusters

Through the performance analysis of Hadoop mobile clusters, this work has focused on two major traffic patterns: 1) between Map and Reduce tasks and 2) between master and slave nodes. It identifies two performance issues that cause frequent performance degradation in mobile distributed analytics with MapReduce. Specifically, overflows of the MAC-layer transmit queue on the slave nodes interrupt long-lived analytical data flows required for data aggregation and overflows of the TCP receive buffer on the master node prevent timely updates of task progress and resource status for job monitoring and scheduling. This chapter introduces adaptive TCP flow control algorithms for improving the performance issues.

5.1 Queueing level control for transmit queue overflow

Most of the previous TCP proposals for both wired and wireless environments (e.g., [24] and [25]) neglect a major influence of the MAC-layer packet loss due to transmit queue overflows. Neither do they consider any round-trip time (RTT) variations due to queueing delays, transmission rate changes, and retransmissions (transmit retries) in the MAC layer while emphasizing the effect of packet loss resulting from network congestion or channel fading. They commonly assume a MAC-layer transmit queue with infinite capacity and zero delay (i.e., no queue dynamics). In practice, the queue's behavior on mobile devices can have a significant impact on TCP performance.

5.1.1 Analysis of MAC-layer transmit queue overflow

Once a TCP connection is established, TCP decides its data rate based on the Transport-layer information only. If the sender's congestion window is not full and the receiver has

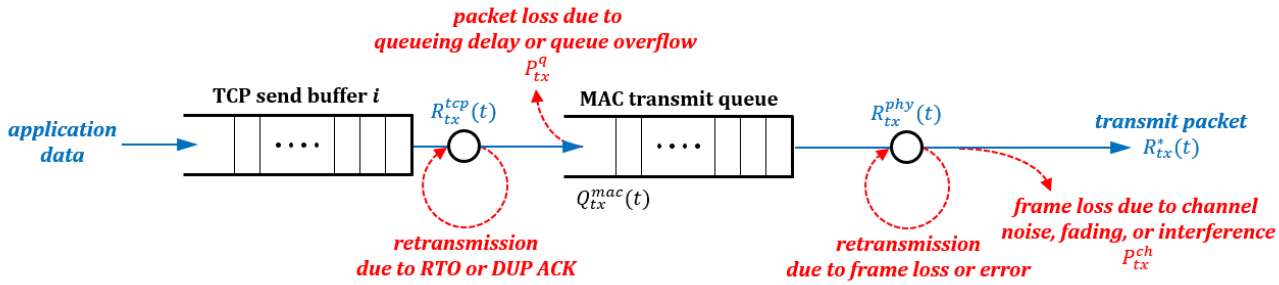


Figure 5.1: TCP packet transmitting process of mobile nodes

a sufficient receive buffer for the connection, TCP packets are continuously injected into a transmit queue of the lower network layer (e.g. MAC-layer transmit queue) until the window size is fully used or there is no more packet in the Transport layer. Meanwhile, the limitations on the TCP performance over mobile devices (e.g., time-varying channel condition, contention-based channel access, PHY transmit rate change, power-saving operation, etc.) can prevent mobile nodes from sending out packets at the same rate as the TCP packet injection (i.e., the inflow of the transmit queue). Thus, the queueing delay for transmission increases and the queueing packets can exceed the limited capacity of the transmit queue. With a simple queue management algorithm (e.g., Tail drop), when the queue is filled to its maximum capacity, the newly arriving packets are dropped until the queue has enough space to accept incoming packets. The loss of packets causes the TCP sender to move into congestion avoidance phases, which reduces throughput in the TCP session.

Figure 5.1 illustrates how mobile nodes process TCP transmitting packets. Consider a wireless single-hop connection consisting of a set of nodes. When one node transfers large data to the other node using TCP, the Transport-layer TCP of the sender releases packets to the MAC-layer transmit queue at the TCP transmit rate $R_{tx}^{tcp}(t)$ which corresponds to the inflow rate of the queue at time t. If there is an influx of packets in the queue, the MAC protocol sends out packets at the Physical-layer transmit rate $R_{tx}^{phy}(t)$ selected by a rate adaptation algorithm. Table 5.1 presents the input parameters of the TCP transmit analysis.

Table 5.1: Parameters for transmit queue overflow analysis

Notation	Parameter
$R_{tx}^{tcp}(t)$	TCP transmit rate at time t
$R_{tx}^{phy}(t)$	Physical-layer transmit rate at time t
$R_{tx}^{*}(t)$	Actual throughput of wireless link at time t
P_{tx}^{ch}	Probability of channel loss
P_{tx}^{q}	Probability of queueing loss
$Q_{tx}^{mac}(t)$	Size of MAC transmit queue at time t
q_{max}	Maximum queue capacity

Let $R_{tx}^{*}(t)$ be the actual throughput of the wireless link, which is equal to or lower than the available network bandwidth because some damaged or lost frames due to the varying channel condition with a probability P_{tx}^{ch} of packet loss may need to be retransmitted. And let $Q_{tx}^{mac}(t)$ denote the size of the transmit queue (i.e., the number of packets in the queue). Using the notations above, the following differential equation that represents the queue's behavior can be derived at a given time t:

$$\frac{\mathrm{d}Q_{tx}^{mac}(t)}{\mathrm{d}t} = \begin{cases} 0, & \text{if } R_{tx}^{tcp}(t) \leq R_{tx}^{phy}(t) \\ R_{tx}^{tcp}(t) - R_{tx}^{phy}(t), & \text{otherwise} \end{cases} \tag{5.1}$$

Thus, $R_{tx}^{*}(t)$ is given by

$$R_{tx}^{*}(t) = \begin{cases} R_{tx}^{phy}(t) - P_{tx}^{ch} \times R_{tx}^{phy}(t), & \text{if } Q_{tx}^{mac}(t) > 0 \\ R_{tx}^{tcp}(t) - P_{tx}^{ch} \times R_{tx}^{tcp}(t), & \text{if } Q_{tx}^{mac}(t) = 0 \end{cases} \tag{5.2}$$

As $R_{tx}^{tcp}(t)$ is bounded by $R_{tx}^{phy}(t)$ that adapts to the channel condition through rate adaptation, both $R_{tx}^{tcp}(t)$ and $R_{tx}^{phy}(t)$ should converge into $R_{tx}^{*}(t)$ in stable state:

$$R_{tx}^{tcp}(t) \approx R_{tx}^{phy}(t) \approx R_{tx}^{*}(t) \tag{5.3}$$

Table 5.2: Variables for transmit queueing level control algorithm

Identifier	**Variables**
tx_qlen	Size of transmit queue
rx_qlen	Size of receive queue
$fc\_threshold$	Threshold for queue inflow control
$tx\_queueing$	Queueing status (set $true$ when queueing data; otherwise $false$)
tx_bound	Max transmit frames in scheduling
max_frames	Max limit of processing frames in one scheduling
$rx\_pending$	Receive status (set $true$ when receiving data; otherwise $false$)

In reality, there is high probability of packet loss caused by poor quality of wireless channel in the mobile environment. When the rate adaptation algorithm aggressively reduces $R_{tx}^{phy}(t)$ due to a short-term increase of channel loss, TCP reacts to the sudden PHY rate reduction but needs a significant amount of settling time to find out the stable rate $R_{tx}^{*}(t)$ by updating its congestion window corresponding to $R_{tx}^{tcp}(t)$ many times. During this time lag, $R_{tx}^{tcp}(t)$ exceeds $R_{tx}^{phy}(t)$. Thus, the number of packets filling the queue increases:

$$\frac{\mathrm{d}Q_{tx}^{mac}(t)}{\mathrm{d}t} = R_{tx}^{tcp}(t) - R_{tx}^{phy}(t) > 0 \tag{5.4}$$

Moreover, mobile nodes share the same channel based on contention-based access. When the channel is busy as many nodes attempt to communicate, the window of transmit opportunity (i.e., transmit time slot) for each node is narrowing. If such delays in queueing and transmitting last during an interval $(t, t+\delta)$ and the transmit queue has a limit on its capacity q_{max}, the queue overflow that may lead to packet loss from TCP retransmission timeout (RTO) with a probability P_{tx}^{q} will occur as follows:

$$\int_{t}^{t+\delta} \frac{\mathrm{d}Q_{tx}^{mac}(t)}{\mathrm{d}t} = q_{max} \tag{5.5}$$

Algorithm 1: Transmit queueing Level control

```
1  if tx_qlen > 0 then
2      if tx_queueing = true then                          /* TCP flow control */
3          if tx_qlen ≥ fc_threshold then
4              Stop TCP packet injection;
5          end
6      else
7          if tx_qlen < fc_threshold then
8              Restart TCP packet injection;
9          end
10     end
11     if rx_pending = false then                          /* MAC resource scheduling */
12         tx_bound ← max_frames;
13     else
14         if tx_qlen = 0 then
15             tx_bound ← max_frames / 2;
16         end
17     end
18 end
```

5.1.2 Transmit queueing level control algorithm

To avoid the transmit queue overflow, TCP needs to regulate the outflow rate (i.e., inflow rate of the transmit queue) according to the available queue space (i.e., the queueing level) first. As the current network protocol architecture that follows strict layering principles do not provide any interface for coordination, interaction, or optimization between network protocols of different layers, a modification of the existing layered protocols is necessary to implement a cross-layer flow control between the Transport and MAC layer.

In addition, mobile devices are usually optimized to improve receive performance, where more resources are statically assigned to make the processing of receive frames faster. If there is no incoming data frame, however, all available resources can be used to transmit packets for reducing the queueing delay. Even though there are incoming frames, the resources for

processing receive and transmit data can be dynamically scheduled according to the data size to be processed as well.

Algorithm 1 presents the steps of the cross-layer flow control and dynamic resource scheduling for stabilizing the queueing level, which are performed on the TCP sender node. Table 5.2 defines variables for monitoring the queue usage, thresholds for initiating the flow control, and parameters for scheduling network resources. If the transmit queue fills and the queue length exceeds the pre-defined threshold, TCP stops injecting packets into the queue. When the queueing level stabilizes, TCP restarts queueing. Then, it checks the receiving data flows. If there is no overhead in processing incoming frames, it uses more resources for transmitting packets in the queue.

5.2 TCP push flow control for receive buffer overflow

The previous research has mostly focused on the partition and aggregation pattern of distributed systems in data center networks, where the input data is partitioned and sent to worker nodes, and the output data of each worker node is transmitted to an aggregation node, but another traffic pattern between the controller and worker nodes also has a significant impact on the performance of the distributed computing system. Although worker nodes complete assigned all sub-workloads faster, the controller's final response to the user's request of the workload can be delayed if there is a bottleneck in reporting the sub-workload completion and transmitting metadata for the final result to the controller. Thus, the response time is substantially affected by the network performance between the controller and worker nodes as well.

5.2.1 Analysis of TCP receive buffer overflow

Every slave node that composes the Hadoop cluster transmits two types of packets (heartbeat packets for quick status updates and data packets for operational data transfers) to the master node, the only controller of Hadoop framework. For the MapReduce operation,

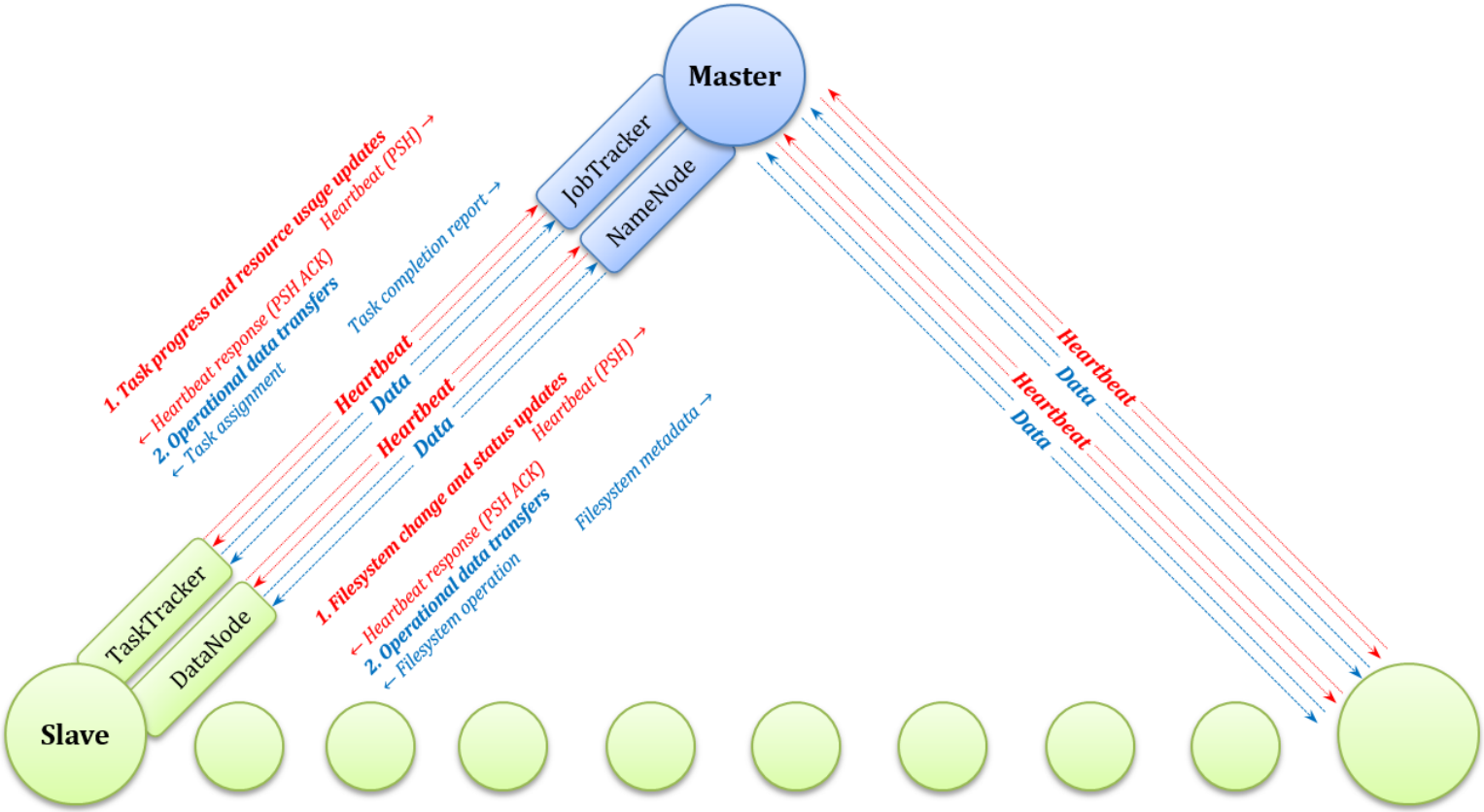
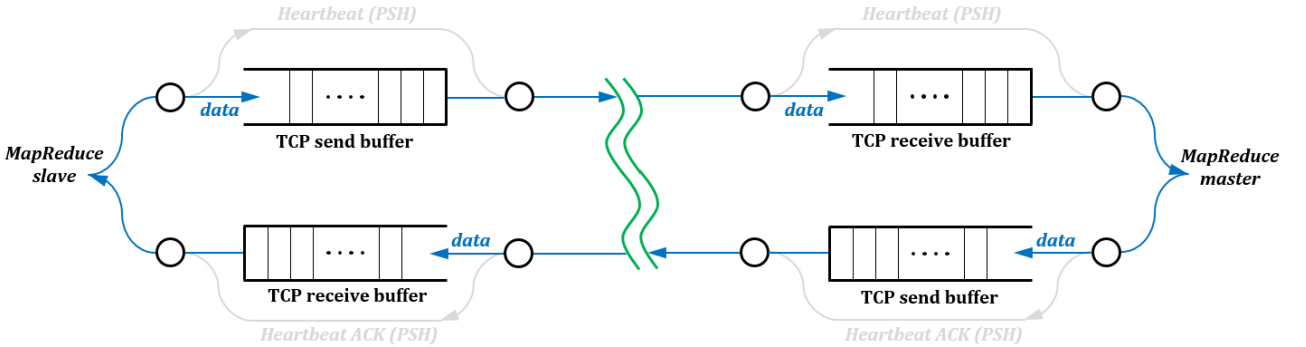


Figure 5.2: Data flows between master and slave nodes

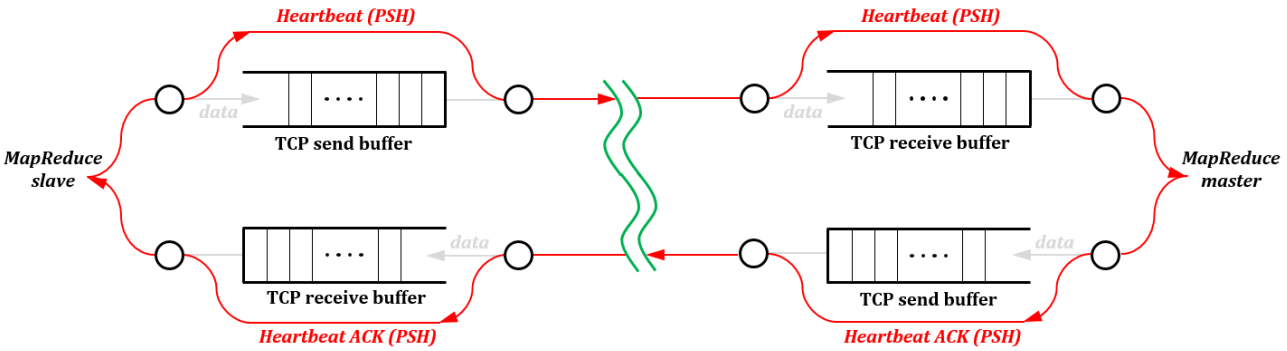
each node runs a simple loop that periodically sends a heartbeat packet to the master. The heartbeat packets contain timely information about the node state (e.g., statistics of resource utilization) and progress of running tasks, which enables the master to monitor task status and restart a failed or slow task rapidly. The master combines these updates to produce a global view of the status of all the jobs being run and their sub-tasks. There are other sporadic operational data transfers for task assignments and task completion reports using data packets in the traffic pattern as well. In addition, each node also communicates with the master using the same kinds of packet for the distributed files system (i.e., HDFS) operation as depicted in Figure 5.2.

The Hadoop software architecture distinctively implements the heartbeat updates using the TCP push function to force an immediate data transfer (without data buffering). When this function is invoked, TCP creates a packet that contains application data, and transmits it right away with the PSH control flag bit set to 1. The PSH flag informs the receiving host that the data should be pushed up to the receiving application immediately. This function is originally designed for time-sensitive TCP connections (e.g., Telnet, HTTP GET request, etc.) and is also useful for TCP-based real-time applications. Figure 5.3 illustrates how the data flow of Hadoop with the normal TCP packets differs from the heartbeat flow with the TCP push packets.

Meanwhile, current window-based TCP congestion/flow control algorithms (e.g., [25], [24]) do not effectively deal with the fast data flow consisting of TCP push packets. Because the TCP push packets unregulated by the typical window-based (or buffer-based) control scheme have a high process priority, a data flow with excessive TCP push packets can exacerbate any congestion level of the TCP connection or processing overhead of inflowing packets stored in TCP receive buffers. For example, the TCP push packets account for about 30 percent of traffic to the master node that coordinates 10 slave nodes in the Hadoop cluster running TeraSort workload with 1GB input data; the master receives around 30 TCP



(a) Data flow



(b) Heartbeat flow

Figure 5.3: Comparison of Hadoop data and heartbeat flow

push packets per second. Thus, the data flows between the master and slave nodes have a non-negligible effect on the performance.

In the performance analysis, the frequent TCP receive buffer overflows on the master node are observed (i.e., the master node advertises TCP zero receive window) while running the Hadoop benchmarks. When a receiver (i.e., master node) is unable to process TCP packets in the receive buffer quickly, the receiver reports decreasing values of the receive window and the sender (i.e., slave node) transmits a portion of data that does not exceed the receive window. If TCP Zero Window is reported by the receiver, the sender immediately stops its transmission of the corresponding TCP session. Thus, every Zero Window advertisement incurs a substantial throughput penalty and transmission delay on the TCP connection due

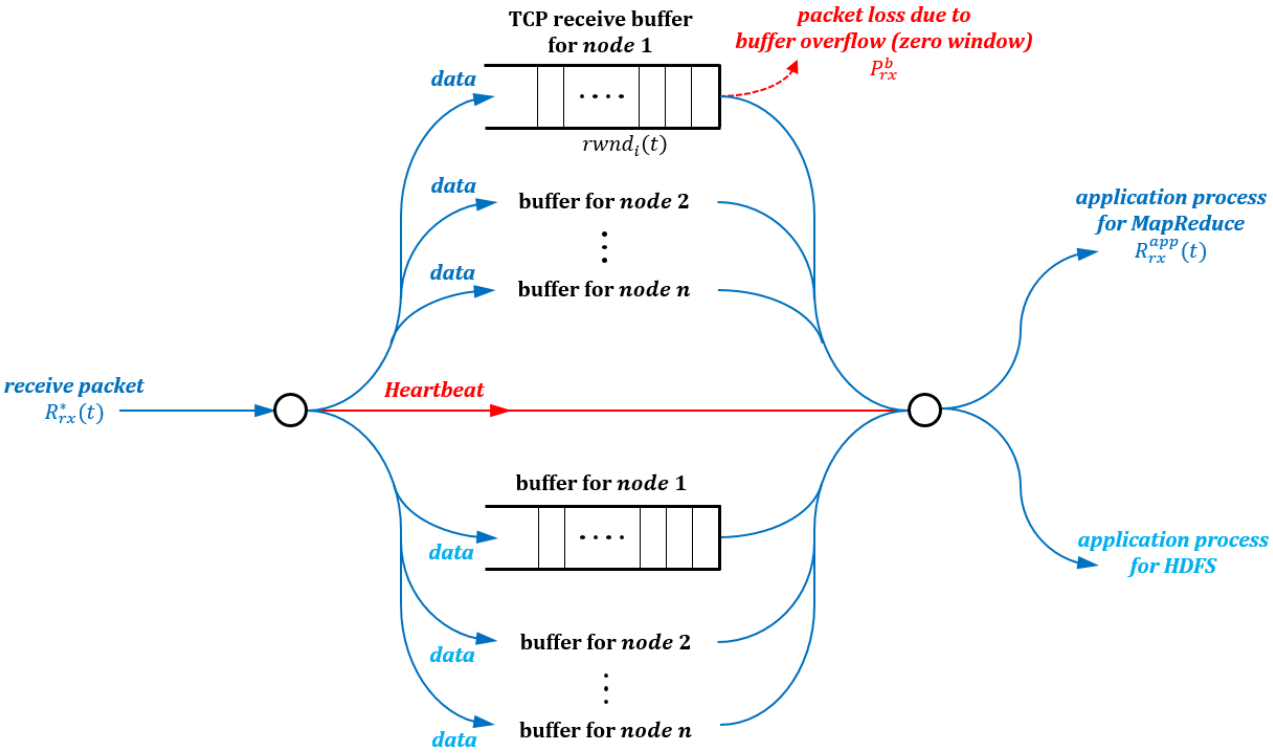


Figure 5.4: TCP packet receiving process of master node

to the significant amount of time taken for the receive window to be re-opened to the normal window size [60].

Figure 5.4 demonstrates how the master node processes TCP receiving packets. Consider one of TCP sessions between the master and slave nodes. The master receives packets from a slave at the rate $R_{rx}^{*}(t)$ and the application of the master processes the packets at the rate $R_{rx}^{app}(t)$. When the master has a bottleneck in processing the receive packets, the TCP receive buffer begins to fill. Let $rwnd_i(t)$ denote the size of the TCP receive buffer i at time t. Table 5.3 presents the input parameters of the TCP receive analysis. The following equation that describes behavior of the receive buffer can be derived at a given time t. If

Table 5.3: Parameters for receive buffer overflow analysis

Notation	Parameter
$R^*_{rx}(t)$	Physical-layer receive rate at time t
$R^{app}_{rx}(t)$	Application's processing rate at time t
P^b_{rx}	Probability of packet loss due to buffer overflow
$rwnd_i(t)$	Size of TCP Receive buffer at time t
$rwnd_{max}$	Maximum buffer capacity

there are no substantial MAC-layer receive buffering delays,

$$\frac{\mathrm{d}rwnd_i(t)}{\mathrm{d}t} = \begin{cases} 0, & \text{if } R^*_{rx}(t) \leq R^{app}_{rx}(t) \\ R^*_{rx}(t) - R^{app}_{rx}(t), & \text{otherwise} \end{cases} \tag{5.6}$$

Since the master node based on a mobile device is subject to resource scarcity on managing its slave nodes by monitoring the job progress and by updating the entire image of the distributed filesystem, there is higher possibility of bottlenecks in processing. For example, the master of Hadoop runs two major controller modules, JobTracker for MapReduce and NameNode for HDFS, and maintains at least $2 \times n$ concurrent receive buffers for TCP connections between the master node with 2 controllers and the n slave nodes (as well as the same number of TCP transmit buffers for bidirectional TCP flows) as depicted in Figure 5.4.

In addition, each slave node periodically sends heartbeats that need to be processed as soon as they arrive. TCP pushes them to the application immediately. The TCP push packets aggravate the process overhead while the master is busy processing many requests. Consequently, $R^*_{rx}(t)$ frequently goes beyond $R^{app}_{rx}(t)$ on the master node. Thus, the receive buffer size (the number of TCP data packets to be processed in the receive buffer) increases:

$$\frac{\mathrm{d}rwnd_i(t)}{\mathrm{d}t} = R^*_{rx}(t) - R^{app}_{rx}(t) > 0 \tag{5.7}$$

When such bottleneck in processing receive packets lasts during the interval $(t, t+\delta)$ and the TCP receive buffer has a limit on its capacity $rwnd_{mac}$, the receive buffer overflow that leads to packet loss from TCP retransmission timeout (RTO) with a probability P_{rx}^{b} may occur as follows:

$$\int_{t}^{t+\delta} \frac{\mathrm{d}rwnd_i(t)}{\mathrm{d}t} = rwnd_{max} \tag{5.8}$$

As a result, the receiver's processing rate $R_{rx}^{app}(t)$ eventually slows down the sender's transmission rate $R_{tx}^{tcp}(t)$ and limits the actual throughput $R_{tx}^{*}(t)$ of the TCP connection. From (3), we have

$$R_{tx}^{tcp}(t) \approx R_{tx}^{phy}(t) \approx R_{tx}^{*}(t) \approx R_{rx}^{*}(t) \approx R_{rx}^{app}(t) \tag{5.9}$$

5.2.2 TCP push flow control algorithm

To avoid the receive buffer overflow on the master node which results in the interruption of receiving the latest status updates and task reports, a modification of the TCP congestion control is needed because the current TCP implementation does not include any performance consideration and sophisticated exception handling for the TCP push function. This work has reviewed the Linux Kernel codes for TCP by focusing on TCP push packet handling and identified a problem in round-trip time (RTT) estimation that is essential for TCP congestion window control and retransmission timeout (RTO) estimation.

The TCP sender measures the RTT of the network path, which includes the propagation delays of the network channels, queueing delays of network devices, and processing delays at the receiver. The TCP implementation estimates the accurate RTT of the path using new RTT samples and updates the RTO based on the estimated RTT. If the RTO is underestimated due to incorrect RTT estimates, the TCP packets can be retransmitted excessively, which also initiates TCP congestion avoidance procedures unnecessarily. Thus, the RTT measure is the most important parameter in the TCP performance.

In the Hadoop cluster, the master node responds (acknowledges) to the heartbeat (TCP push packet) using the same type of packet, i.e., every heartbeat call incurs a bidirectional

Table 5.4: Variables for TCP push flow control algorithm

Identifier	**Variables**
fc_mode	Flow control status (set $true$ when enabled; otherwise $false$)
$zero\_window$	TCP Zero Window reception (set $true$ when advertised by master; otherwise $false$)
psh_acked	TCP push ACK reception (set $true$ when acknowledged; otherwise $false$)
$delayed\_ack$	TCP delayed ACK status (set $true$ when enabled; otherwise $false$)
$cwnd$	TCP congestion window
$packets\_in\_flight$	Unacknowledged TCP data in flight

TCP push stream. As the TCP push packet is passed directly to the application and its acknowledgment is transmitted without buffering on the receiver, the TCP push stream has a shorter processing delay compared to the normal TCP data stream as shown in Figure 5.3. If the short RTT of the push stream is frequently used for the samples of RTT estimation, excessive retransmissions of many normal TCP packets will occur after the RTO is underestimated according to the incorrect RTT estimates. Also, the fast acknowledgements of the push packets increase the sender's congestion window quickly. As a result, the sender (i.e., slave node) transmits too many packets that overwhelm the available network bandwidth and/or the receiver (i.e., master node)'s processing capability.

Algorithm 2 includes the TCP flow control procedure for improving receive buffer overflows of the master node, which is performed on the slave nodes. Table 5.4 defines variables for monitoring the Zero Window events, thresholds for initiating the flow control, and parameters for the TCP control. If Zero Window is advertised by the master node, the slave node initiates the TCP push flow control mode. In this mode, the TCP control skips the specific receive steps for the RTT update and RTO estimation using the RTT sample of TCP push streams; also bypasses the step for delayed ACK timeout (ATO) estimation if enabled. Then, it moderates the congestion window by taking the smallest size of the congestion window and the unacknowledged data (i.e., the actual bandwidth limit of the connection) in order to prevent an outburst from the fast growth due to the quick acknowledgements of

Algorithm 2: TCP push flow control

```
1  if zero_window = true then                          /* Initiate flow control */
2  |   fc_mode ← true;
3  end
4  if fc_mode = true then                              /* TCP push flow control */
5  |   if psh_acked = true then
6  |   |   Skip RTT update;
7  |   |   Skip RTO estimation;
8  |   |   if delayed_ack = true then
9  |   |   |   Skip ATO estimation;
10 |   |   end
11 |   |   cwnd ← min(cwnd, packets_in_flight);
12 |   end
13 end
```

the push packets. This algorithm is implemented in the TCP protocol of the experimental platform.

Chapter 6

Evaluation of proposed solutions

This chapter presents performance evaluation of two proposed solutions: the transmit queueing level control algorithm for avoiding MAC-layer transmit queue overflows on the slave nodes and TCP push flow control algorithm for mitigating TCP receive buffer overflows on the master node.

6.1 Transmit queueing level control algorithm

To avoid frequent MAC-layer transmit queue overflows on the mobile cluster nodes, which interrupt long-lived analytical data flows required for the partition and aggregation workflow of distributed processing frameworks, Algorithm 1 performed on the TCP sender includes the cross-layer flow control for TCP packet injection and dynamic network resource scheduling for incoming and outgoing frames to minimize the queueing delay and stabilizing the queueing level. The algorithm is implemented in the TCP and MAC protocol of Linux-based Android OS kernel on the experimental mobile platform (Google Nexus 7).

6.1.1 Performance improvement of peer-to-peer data transfer

First, TCP communication performance between two mobile devices was measured in order to clarify the cause of performance degradation during large data transfers. One device was configured as a transmitter and the other as a receiver. A network performance measurement tool, Iperf, was employed to investigate communication performance, which generates constant TCP traffic. In addition, several scripts to control continuous Iperf test runs and collect performance measurements (e.g., TCP throughput) by configuring the Iperf test parameters according to each experimental setup were developed. The scripts

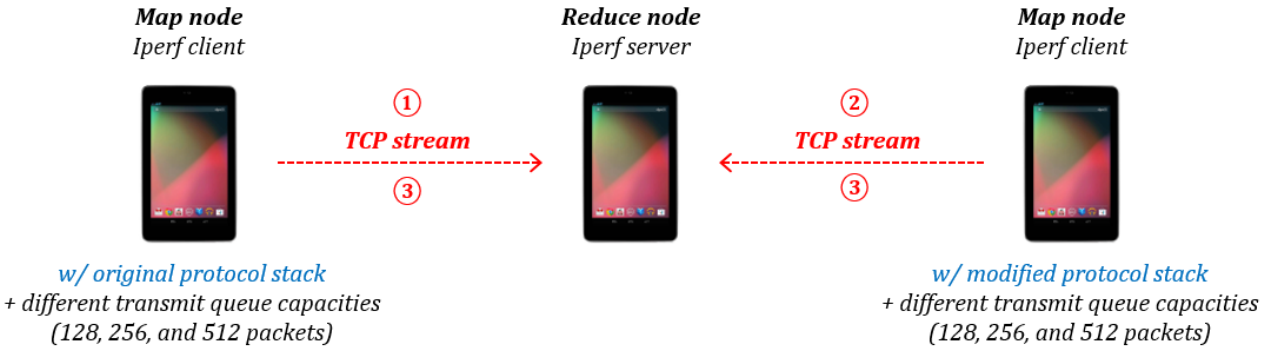
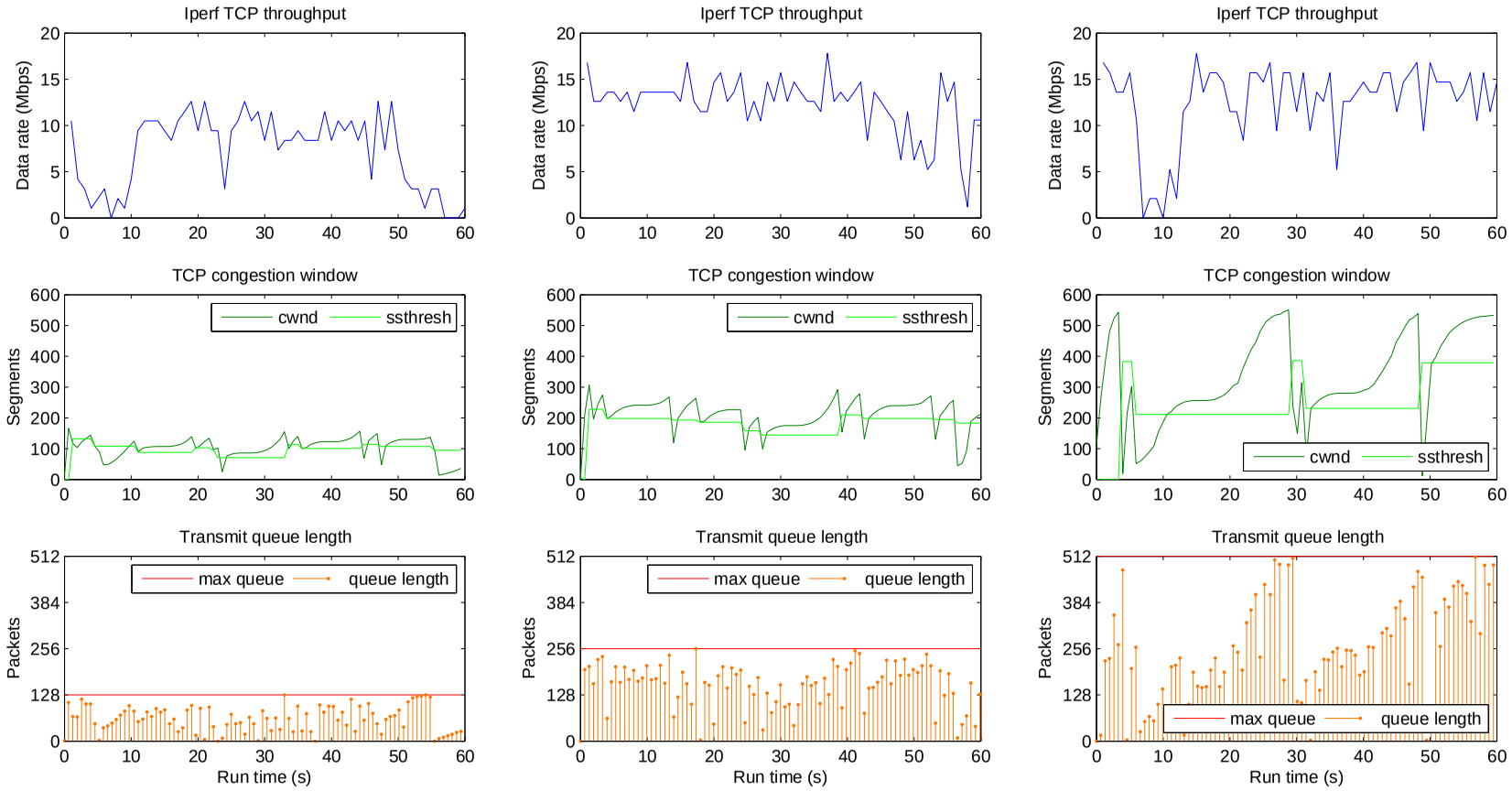


Figure 6.1: Evaluation setup for transmit queueing level control

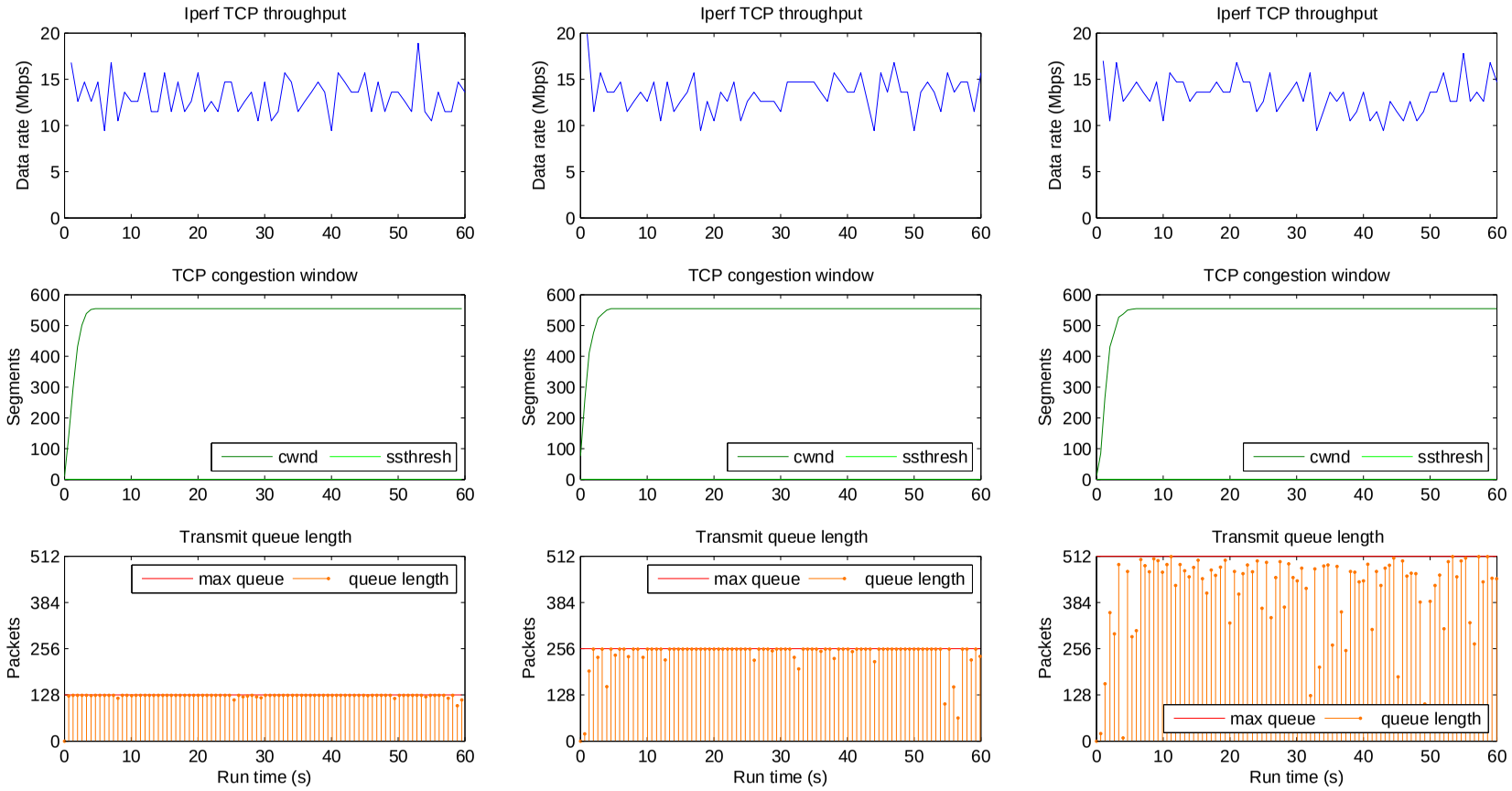
also collected TCP and link statistics (e.g., TCP congestion window, transmit queue size, transmission rate, retransmission, and transmission failure) from the Linux kernel and its wireless driver. As the performance of a wireless link is usually affected by the surrounding channel conditions, the experimental results may exhibit variations. To eliminate this effect, the communication performance was measured at 10 feet range with an unobstructed line-of-sight, where all measurements reported negligible differences. Figure 6.1 presents the evaluation setup for the transmit queueing level control.

Figure 6.2 indicates how the frequent transmit queue overflows affect the TCP performance between two mobile nodes. Three different queue capacities of the transmitter with the original protocol stack are tested; the smaller is 128 packets, the default 256 packets, and the larger 512 packets. It is found that the queue overflows are even observed in the larger queue and they cause frequent TCP throughput collapse with constant adjustments of TCP congestion window from the transmit queueing losses. Figure 6.3 also demonstrates how the proposed algorithm improves the TCP performance. The same queue sizes of the transmitter with the modified protocol stack are applied. As the algorithm implements the TCP flow control and dynamic resource scheduling for reliable data communications, two mobile nodes achieve better TCP throughput with the stable congestion window in all the different queue capacities. Note that the transmit queue size should be optimized considering



(a) Max queue: 128 pkts (b) Max queue: 256 pkts (c) Max queue: 512 pkts

Figure 6.2: TCP performance of mobile nodes with different MAC queue sizes



(a) Max queue: 128 pkts (b) Max queue: 256 pkts (c) Max queue: 512 pkts

Figure 6.3: TCP performance with modified network stack

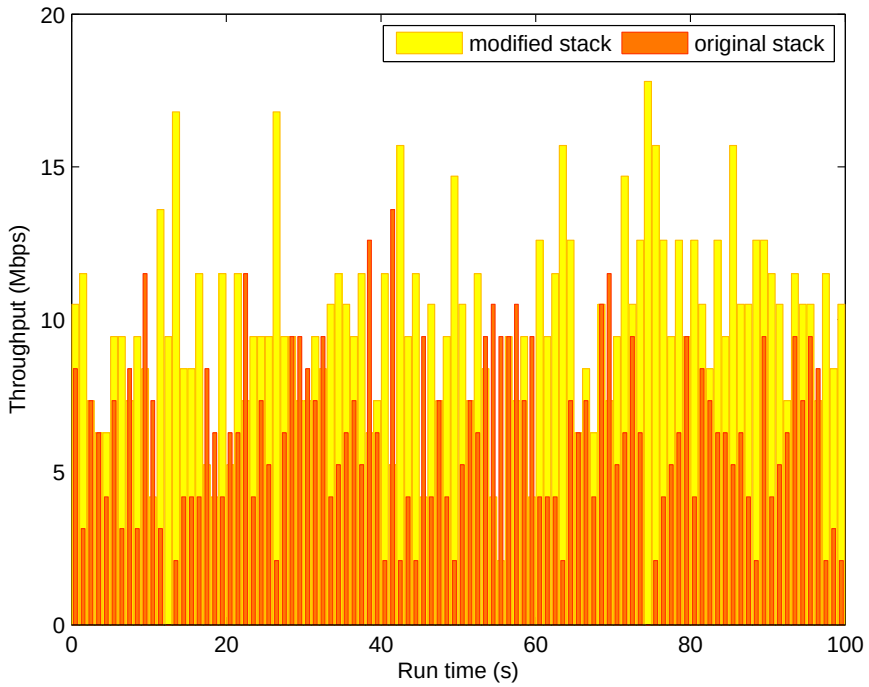


Figure 6.4: TCP throughput of original and modified network stack

other communication protocol usages (e.g., UDP) even though the small queue size works well with the adaptive TCP flow control algorithm.

6.1.2 Performance improvement in data aggregation

Next, three mobile devices were configured to generate aggregate data traffic that is similar to the data flow pattern between the Map and Reduce nodes; see Figure 6.1. Two devices simultaneously transmit a constant TCP stream to one device. To compare the TCP performance between the original and modified protocol stack, one transmitter was configured with the original stack and the other with the modified stack that implements the algorithm. The TCP performance is measured in the same way as the prior experiments using the Iperf scripts.

Figure 6.4 shows a significant improvement in TCP performance. During the 100-second test with the original stack, the zero throughput due to the transmit queue overflows

is observed twice at 13 and 75 second. On the contrary, the transmitter with the modified stack sends more TCP data reliably without throughput collapse as indicated in the figure and its average throughput (9.82 Mbps) is significantly improved compared to that of the original one (6.18 Mbps).

Although the proposed algorithm improves the performance problem, the critical values for the flow control threshold (or transmit queue size) and resource scheduling parameters should be carefully determined in consideration of other performance factors (e.g., memory utilization, network condition, and other communication protocols) and the operating conditions (or environments). Thus, the future work should continue to refine the algorithms by validating through extensive experiments using various benchmarks and sample applications.

6.2 TCP push flow control algorithm

In order to prevent the TCP receive buffer overflows on the master node due to the use of TCP push packets, which results in interruptions in the reception of the latest status updates and task reports from slave nodes, Algorithm 2 performed on slave nodes includes the TCP push flow control for mitigating the side effects of TCP push flows that cause excessive transmissions from fast congestion window growth and recurrent RTO underestimation. It moderates the congestion window outburst and bypasses the receive steps for the RTO estimation using RTT updates from the TCP push streams when the buffer overflow is detected. The algorithm is implemented in the Linux TCP protocol of the experimental Android platform.

6.2.1 Packet analysis of Hadoop master running TeraSort

For evaluation of the algorithm, the TeraSort benchmark with 1 GB input data is executed in the same configuration of the previous performance analysis, where the mobile cluster that runs the Hadoop software consists of a single master node and ten slave nodes

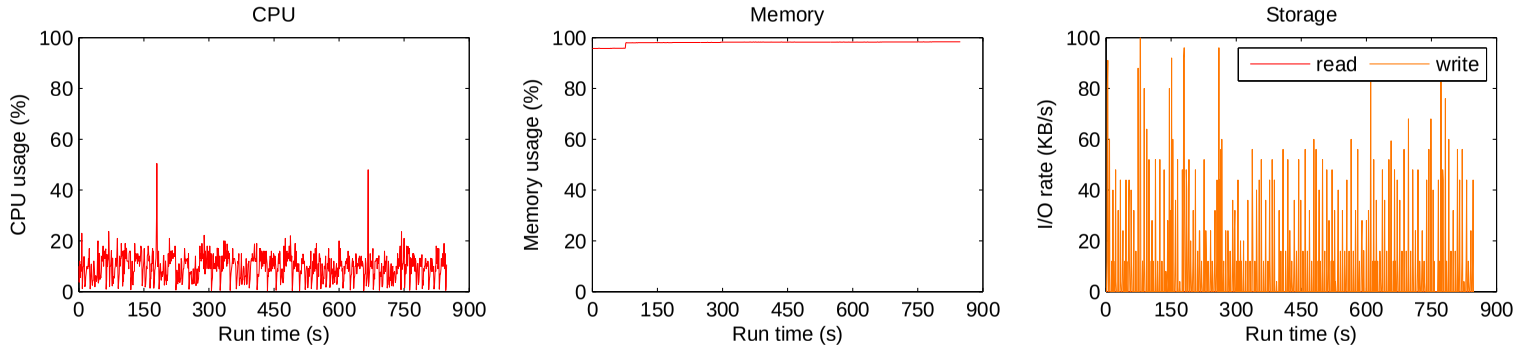


Figure 6.5: Resource utilization of master node running TeraSort workload

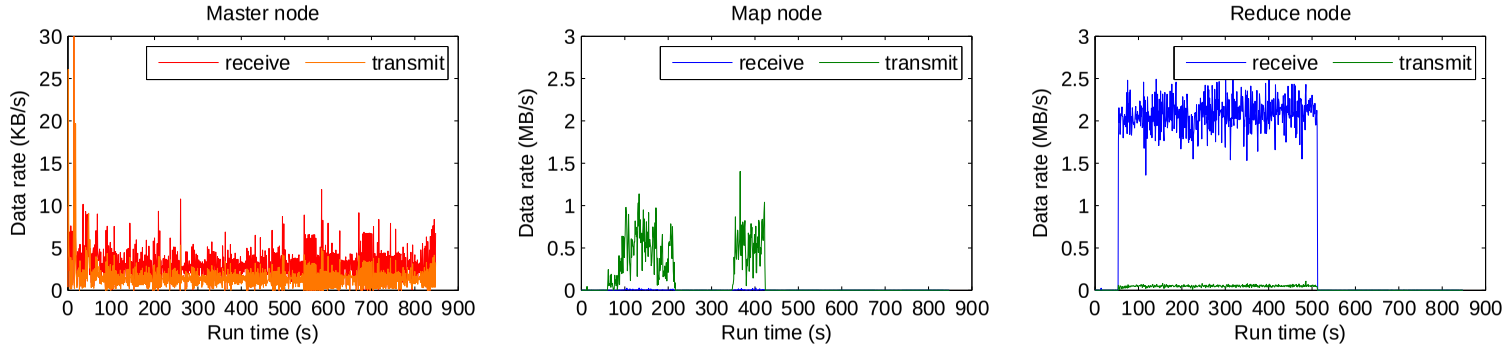


Figure 6.6: Traffic patterns of cluster nodes running TeraSort workload

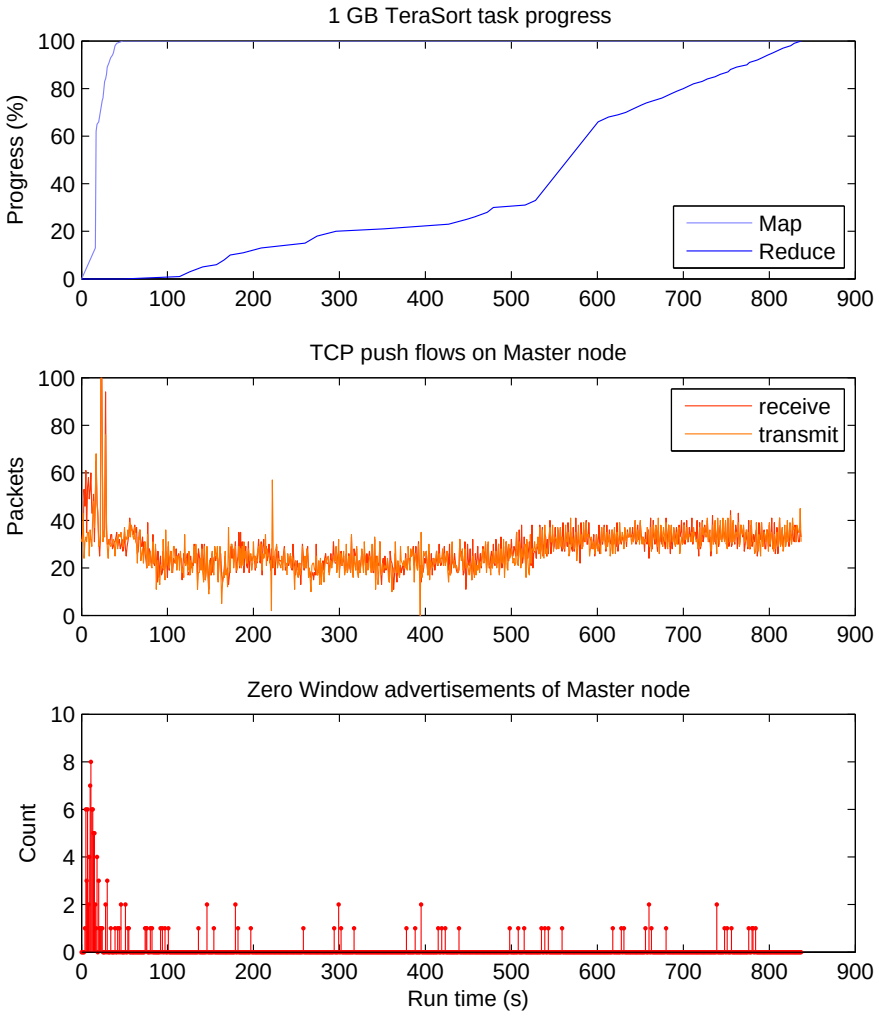


Figure 6.7: TCP packet analysis of master node

configured with the default values for system parameters of Android OS and Hadoop framework. The usages of computing and networking resources of each node are monitored with the performance monitoring tool during the experiments.

First, slave nodes with the original TCP protocol are used for the mobile cluster in order to investigate master node's behavior under the CPU and I/O bound TeraSort testing. As the master node based on a mobile device is subject to resource scarcity on managing all the cluster nodes by monitoring their task progress with resource usages and by updating the entire image of the distributed filesystem, the high memory usage and steady storage

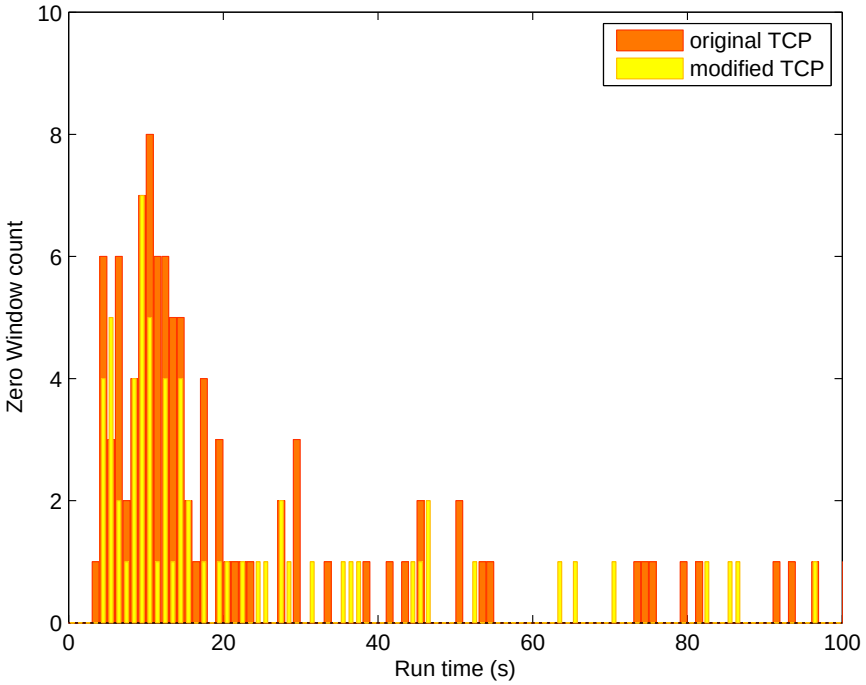


Figure 6.8: Zero Window advertisements of original and modified TCP

utilization of the master node are commonly observed as shown in Figure 6.5. Figure 6.6 also displays the master's incessant sort-lived traffic pattern (i.e., TCP push flows) compared to slave nodes running Map and Reduce tasks during the experiment. In addition, Figure 6.7 includes the TCP packet analysis of the master node with the task progress, which indicates that the master node processes around 30 incoming and outgoing TCP push packets (i.e., heartbeats and acknowledgements) every second. As a result, the master node receives too many heartbeat and data packets that overwhelm its processing capability at some time and advertises frequent Zero Window to halt the TCP transmission; see Figure 6.7.

6.2.2 Performance improvement of TeraSort workload

Next, the slave nodes where the modified TCP protocol is implemented are deployed on the mobile cluster in order to evaluate the proposed algorithm. Since it prevents the RTT update and RTO estimation using the RTT sample of TCP push streams when the

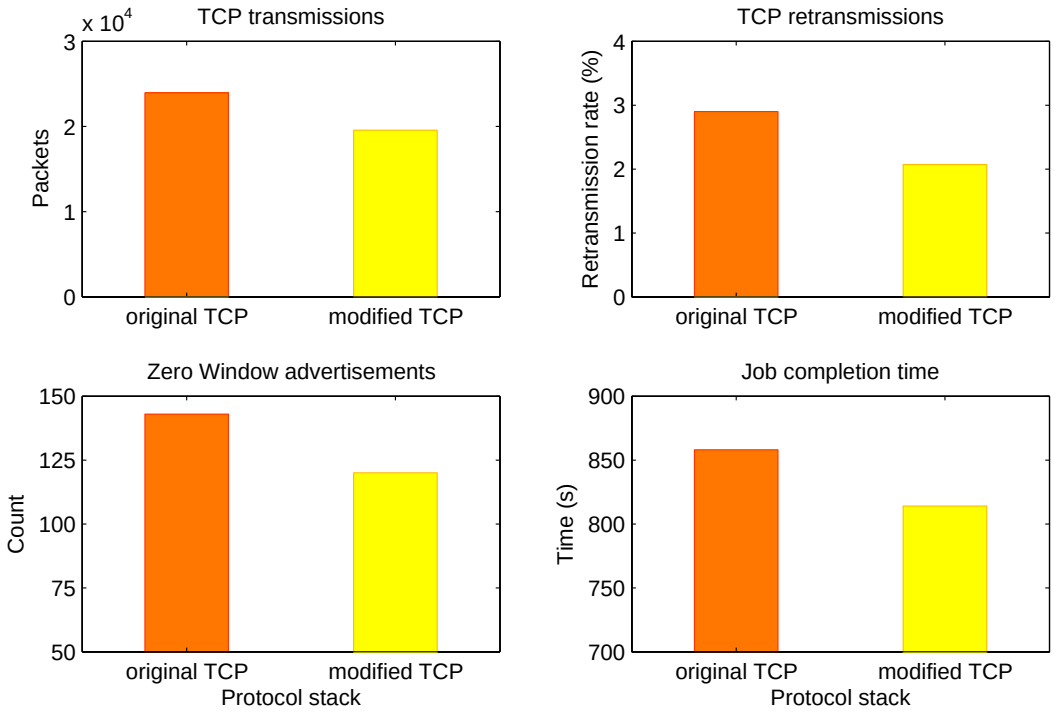


Figure 6.9: Performance improvement of experimental mobile cluster

master node advertises Zero Window and moderates the TCP congestion window to avoid an outburst from the fast growth due to the quick acknowledgements of the push packets, there are marked decreases in the Zero Window advertisement (i.e., receive buffer overflow occurrence) of the master node as shown in Figure 6.8. When all the cluster nodes have the implementation of the TCP push flow control algorithm along with the transmit queueing level control algorithm, it is observed that the TCP retransmission rate of the slave nodes (as well as the total number of transmit packets) can be reduced and the overall job completion time also expects to be shortened; see Figure 6.9.

Even though the proposed algorithm mitigates the performance problem, there are still a significant number of buffer overflow occurrences on the master node. Therefore, the future work should continue to improve the proposed algorithm. For example, adaptive buffer management for many concurrent receive buffers (e.g., twice the number of slaves

for Hadoop master) is required to receive more incoming packets when considering the slow application processing rate under resource scarcity of mobile devices.

As analyzed in this study, frequent overflows of the MAC-layer transmit queue and TCP receive buffer on the mobile cluster nodes are the two most significant performance issues, which interrupt reliable analytical data interchanges for mobile distributed analytics. Since the higher frequency of the overflows leads to a large number of under-utilization periods and fluctuations in the TCP performance, for example, the Hadoop mobile clusters may not able to complete a submitted job within an expected time frame by reporting a significant number of response timeout in the task running. Hence, a better TCP flow control algorithm for mobile distributed analytic applications should be proposed in order to improve the performance problems and achieve the desired performance over variable mobile environments and critical performance limitations.

Chapter 7

Conclusion

This chapter concludes this dissertation with a summary of the previous chapters and a discussion of main contributions and further research suggestions.

7.1 Summary

In this work, the benefits and challenges of employing practical mobile devices for implementing mobile ad hoc cloud were analyzed. The current trends of mobile technologies and mobile services were presented with the different concepts of mobile cloud computing in Chapter 1. The overviews of Hadoop distributed analytic framework and TCP flow control for performance analysis were introduced in Chapter 2 and the prior efforts to develop mobile cloud services using mobile devices were discussed with other useful techniques and approaches therein. In order to understand performance issues of mobile distributed analytic applications, the experimental performance studies were conducted by developing a test bed of mobile cloud clusters with Hadoop analytic framework in Chapter 3 and furthermore the simulation studies were conducted by developing the MapReduce simulator based on a network simulator to address extensive mobility scenarios and operating setups.

Moreover, this work identified performance issues through performance analysis. Despite advances in mobile technologies, mobile devices still contains significant imperfections in transmitting and receiving reliable data streams required to avoid any interruptions while performing distributed processing, which come from limitations on TCP performance over wireless networks and problems of using mobile devices with resource constraints. The problem statements and the main goal of this study can be found in Chapter 4 where the research questions were presented. In order to improve the performance issues, this work proposed

two adaptive TCP flow control algorithms for mitigating the transmit queue overflows and the receive buffer overflows of mobile devices with the analysis of TCP performance problems over Hadoop MapReduce framework in Chapter 5. Finally, the presented algorithms were evaluated in Chapter 6 and it was shown that the proposed solutions improve the TCP performance and shorten the overall MapReduce processing time.

7.2 Discussion

The main contributions of the present work can be summarized as follows. First, this work discussed the benefits and the challenges of exploiting mobile devices for distributed cloud applications, showing its feasibility with Hadoop analytic framework, and also investigates critical performance issues for reliable data communications between mobile devices in the workflow of distributed computing. Second, unlike earlier approaches this work examined the performance of Hadoop mobile clusters by performing extensive experiments using typical Hadoop benchmarks representing CPU, memory and/or I/O intensive applications. The newest release of Hadoop software framework with its enhancements was entirely ported to the latest Android-based mobile devices through the mobile virtualization. Third, this work developed the MapReduce simulator based on the ns-2 network simulator in order to comprehensively evaluate the performance and efficiency of mobile cloud clusters in extensive operating environments, which allowed it to identify more performance issues under different cluster (or workload) scales, dynamic node mobility, and various wireless channel conditions. Lastly, this work analyzed the TCP performance problems resulting from distinct traffic patterns of MapReduce-based Hadoop distributed framework and proposed adaptive TCP flow control algorithms for improving the performance of mobile cloud clusters by mitigating the effects of the frequent transmit queue and receive buffer overflows of mobile devices.

In order to improve the presented algorithms for the reliable data communications between mobile cluster nodes, the future work should enable the algorithms to dynamically adjust the values of the performance parameters for TCP flow control (e.g., threshold for

queue inflow control) and network resource scheduling (e.g., number of transmit frames in scheduling) according to the operating conditions although the current algorithms with predefined values mitigated the performance problems in the experimental setups. In addition, an adaptive buffer management technique for many concurrent TCP receive buffers of the controller node is necessary to accommodate more incoming packets considering the slow application processing rate of mobile devices with small buffer capacity. It is also important that the algorithms should be refined, validating through extensive experiments and simulations using other distributed analytic frameworks and various real-world applications.

While this work mostly focused on the performance of practical distributed analytic applications on mobile cloud clusters in terms of job processing time (or completion time) and response time, some other studies have paid attention to mobile node's energy efficiency that is another key performance factor for enabling data analysis and mining over mobile devices. Reducing energy consumption is one of the most important design aspects for small form-factor mobile platforms, such as smartphones and tablets. Therefore, an energy-aware scheduling over the mobile cluster for optimizing energy utilization of cluster nodes should be taken into account for reliable mobile distributed analytics.

Bibliography

[1] "IDC Predictions 2014: Battles for Dominance and Survival on the 3rd Platform," Research, IDC, Inc., 2013. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=244606

[2] "Forecast: PCs, Ultramobiles and Mobile Phones, Worldwide, 2010-2017, 4Q13 Update," Research, Gartner, Inc., 2014. [Online]. Available: https://www.gartner.com/doc/2639615

[3] "Bring Your Own Device," Website, The White House, Aug. 2012. [Online]. Available: http://www.whitehouse.gov/digitalgov/bring-your-own-device

[4] PassMark Software, "CPU Benchmarks," 2014. [Online]. Available: http://www.cpubenchmark.net/

[5] "Worldwide Mobile Worker Population 2011-2015 Forecast," Research, IDC, Inc., Dec. 2011. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=238366#.USZZJ6WNEwE

[6] "Top 10 Strategic Technology Trends for 2013," Research, Gartner, Inc., 2012. [Online]. Available: http://www.gartner.com/technology/research/top-10-technology-trends

[7] Dinh, Hoang T and Lee, Chonho and Niyato, Dusit and Wang, Ping, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, 2011. [Online]. Available: http://dx.doi.org/10.1002/wcm.1203

[8] Fernando, Niroshinie and Loke, Seng W and Rahayu, Wenny, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84 – 106, 2012.

[9] Satyanarayanan, Mahadev and Bahl, Paramvir and Caceres, Ramón and Davies, Nigel, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[10] Zachariadis, Stefanos and Mascolo, Cecilia and Emmerich, Wolfgang, "Satin: A Component Model for Mobile Self Organisation," in *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*. Springer Berlin Heidelberg, 2004, pp. 1303–1321.

[11] Wikipedia, "Cloud computing," 2014. [Online]. Available: http://en.wikipedia.org/wiki/Cloud_computing

[12] "Gartner Identifies the Top 10 Strategic Technology Trends for 2014," Press Release, Gartner, Inc., 2012. [Online]. Available: http://www.gartner.com/newsroom/id/2603623

[13] "Hadoop project," Website, Apache Software Foundation. [Online]. Available: http://hadoop.apache.org

[14] Dean, Jeffrey and Ghemawat, Sanjay, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[15] Ghemawat, Sanjay and Gobioff, Howard and Leung, Shun-Tak, "The Google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[16] White, Tom, *Hadoop: The definitive guide*. O'Reilly Media, 2012.

[17] Rajaraman, Anand and Ullman, Jeffrey David, *Mining of Massive Datasets*. Cambridge University Press, 2012.

[18] "Big Data in the Enterprise: Network Design Considerations," White paper, Cisco Systems, Inc., 2011. [Online]. Available: http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9670/white_paper_c11-690561.html

[19] Yang, Hailong and Luan, Zhongzhi and Li, Wenjun and Qian, Depei, "MapReduce workload modeling with statistical approach," *Journal of Grid Computing*, vol. 10, no. 2, pp. 279–310, 2012.

[20] Afanasyev, Alexander and Tilley, Neil and Reiher, Peter and Kleinrock, Leonard, "Host-to-Host Congestion Control for TCP," *Communications Surveys & Tutorials, IEEE*, vol. 12, no. 3, pp. 304–342, 2010.

[21] Postel, Jon, "Transmission control protocol, RFC 793," 1981.

[22] Jacobson, Van, "Congestion avoidance and control," in *ACM SIGCOMM Computer Communication Review*, vol. 18, no. 4. ACM, 1988, pp. 314–329.

[23] Karn, Phil and Partridge, Craig, "Improving round-trip time estimates in reliable transport protocols," *ACM SIGCOMM Computer Communication Review*, vol. 17, no. 5, pp. 2–7, 1987.

[24] Grieco, Luigi A and Mascolo, Saverio, "Performance Evaluation and Comparison of Westwood+, New Reno, and Vegas TCP Congestion Control," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 25–38, 2004.

[25] Ha, Sangtae and Rhee, Injong and Xu, Lisong, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[26] Vaquero, Luis M and Rodero-Merino, Luis and Caceres, Juan and Lindner, Maik, "A Break in the Clouds: Towards a Cloud Definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008.

[27] Armbrust, Michael and Fox, Armando and Griffith, Rean and Joseph, Anthony D and Katz, Randy and Konwinski, Andy and Lee, Gunho and Patterson, David and Rabkin, Ariel and Stoica, Ion and others, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[28] Buyya, Rajkumar and Yeo, Chee Shin and Venugopal, Srikumar and Broberg, James and Brandic, Ivona, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[29] Zhang, Qi and Cheng, Lu and Boutaba, Raouf, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[30] Mei, Lijun and Chan, Wing Kwong and Tse, T.H., "A Tale of Clouds: Paradigm Comparisons and Some Thoughts on Research Issues," in *The 3rd IEEE Asia-Pacific Services Computing Conference (APSCC) 2008*. IEEE, 2008, pp. 464–469.

[31] Wang, Lizhe and Von Laszewski, Gregor and Younge, Andrew and He, Xi and Kunze, Marcel and Tao, Jie and Fu, Cheng, "Cloud Computing: a Perspective Study," *New Generation Computing*, vol. 28, no. 2, pp. 137–146, 2010.

[32] Marinelli, Eugene E, "Hyrax: Cloud Computing on Mobile Devices using MapReduce," Master's thesis, Carnegie Mellon University, 2009.

[33] Teo, Chye Liang Vincent, "Hyrax: Crowdsourcing Mobile Devices to Develop Proximity-Based Mobile Clouds," Master's thesis, Carnegie Mellon University, 2012.

[34] Huerta-Canepa, Gonzalo and Lee, Dongman, "A Virtual Cloud Computing Provider for Mobile Devices," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*. ACM, 2010, p. 6.

[35] Shi, Cong and Lakafosis, Vasileios and Ammar, Mostafa H and Zegura, Ellen W, "Serendipity: Enabling Remote Computing among Intermittently Connected Mobile Devices," in *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*. ACM, 2012, pp. 145–154.

[36] Shi, Cong and Ammar, Mostafa H and Zegura, Ellen W and Naik, Mayur, "Computing in Cirrus Clouds: The Challenge of Intermittent Connectivity," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 23–28.

[37] Kemp, Roelof and Palmer, Nicholas and Kielmann, Thilo and Bal, Henri, "Cuckoo: A Computation Offloading Framework for Smartphones," *Mobile Computing, Applications, and Services*, pp. 59–79, 2012.

[38] Lee, Seungbae and Grover, Kanika and Lim, Alvin, "Enabling actionable analytics for mobile devices: performance issues of distributed analytics on Hadoop mobile clusters," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 2, no. 1, p. 15, 2013.

[39] Chen, Yanpei and Griffit, Rean and Zats, David and Katz, Randy H, "Understanding TCP Incast and Its Implications for Big Data Workloads," University of California at Berkeley, Tech. Rep., 2012.

[40] Alizadeh, Mohammad and Greenberg, Albert and Maltz, David A and Padhye, Jitendra and Patel, Parveen and Prabhakar, Balaji and Sengupta, Sudipta and Sridharan, Murari, "Data center tcp (dctcp)," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 63–74, 2010.

[41] Chen, Yanpei and Griffith, Rean and Liu, Junda and Katz, Randy H and Joseph, Anthony D, "Understanding TCP incast throughput collapse in datacenter networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 73–82.

[42] Phanishayee, Amar and Krevat, Elie and Vasudevan, Vijay and Andersen, David G and Ganger, Gregory R and Gibson, Garth A and Seshan, Srinivasan, "Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems," in *FAST*, vol. 8, 2008, pp. 1–14.

[43] Vasudevan, Vijay and Phanishayee, Amar and Shah, Hiral and Krevat, Elie and Andersen, David G and Ganger, Gregory R and Gibson, Garth A and Mueller, Brian, "Safe and effective fine-grained TCP retransmissions for datacenter communication," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4. ACM, 2009, pp. 303–314.

[44] Wu, Haitao and Feng, Zhenqian and Guo, Chuanxiong and Zhang, Yongguang, "ICTCP: Incast Congestion Control for TCP in data center networks," in *Proceedings of the 6th International COnference*. ACM, 2010, p. 13.

[45] Alizadeh, Mohammad and Atikoglu, Berk and Kabbani, Abdul and Lakshmikantha, Ashvin and Pan, Rong and Prabhakar, Balaji and Seaman, Mick, "Data center transport mechanisms: Congestion control theory and IEEE standardization," in *2008 46th Annual Allerton Conference on Communication, Control, and Computing*. IEEE, 2008, pp. 1270–1277.

[46] Krevat, Elie and Vasudevan, Vijay and Phanishayee, Amar and Andersen, David G and Ganger, Gregory R and Gibson, Garth A and Seshan, Srinivasan, "On application-level approaches to avoiding TCP throughput collapse in cluster-based storage systems," in *Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07*. ACM, 2007, pp. 1–4.

[47] "Mumak: Map-Reduce Simulator," Website, Apache Software Foundation. [Online]. Available: https://issues.apache.org/jira/browse/MAPREDUCE-728

[48] Hammoud, Suhel and Li, Maozhen and Liu, Yang and Alham, Nasullah Khalid and Liu, Zelong, "MRSim: A discrete event based MapReduce simulator," in *Seventh International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, vol. 6. IEEE, 2010, pp. 2993–2997.

[49] Wang, Guanying, "Evaluating MapReduce System Performance: A Simulation Approach," Ph.D. dissertation, Virginia Polytechnic Institute and State University, 2012.

[50] "The Network Simulator - ns," Website, Nanam. [Online]. Available: http://nsnam.isi.edu/nsnam/index.php/Main_Page

[51] Huang, Shengsheng and Huang, Jie and Dai, Jinquan and Xie, Tao and Huang, Bo, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," in *IEEE 26th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 2010, pp. 41–51.

[52] "NEXUS 7," Website, Google, Inc., 2012. [Online]. Available: http://www.google.com/nexus/7

[53] "IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC)and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput," *IEEE Std 802.11n-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009)*, pp. 1–565, 2009.

[54] Enck, William and Octeau, Damien and McDaniel, Patrick and Chaudhuri, Swarat, "A Study of Android Application Security," in *USENIX security symposium*, 2011.

[55] "LinuxonAndroid project," Website. [Online]. Available: http://linuxonandroid.org

[56] Wikipedia, "Mobile virtualization," 2014. [Online]. Available: http://en.wikipedia.org/wiki/Mobile_virtualization

[57] Kebarighotbi, Ali and Cassandras, Christos G, "Timeout Control in Distributed Systems using Perturbation Analysis," in *2011 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*. IEEE, 2011, pp. 5437–5442.

[58] "IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pp. 1–2793, 2012.

[59] Khademi, Naeem and Welzl, Michael and Gjessing, Stein, "Experimental evaluation of TCP performance in multi-rate 802.11 WLANs," in *2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 2012, pp. 1–9.

[60] Sanadhya, Shruti and Sivakumar, Raghupathy, "Adaptive Flow Control for TCP on Mobile Phones," in *2011 Proceedings IEEE INFOCOM*. IEEE, 2011, pp. 2912–2920.