Handbook of Research on Cloud Infrastructures for Big Data Analytics

Pethuru Raj
IBM India Pvt Ltd, India

Ganesh Chandra Deka
Ministry of Labour and Employment, India

A volume in the Advances in Data Mining and Database Management (ADMDM) Book Series

Information Science REFERENCE
An Imprint of IGI Global

Managing Director:	Lindsay Johnston
Production Editor:	Jennifer Yoder
Development Editor:	Austin DeMarco
Acquisitions Editor:	Kayla Wolfe
Typesetter:	Michael Brehm
Cover Design:	Jason Mull

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

CIP Data (Pending)

978-1-4666-5864-6 (ISBN Hardcover)
978-1-4666-5865-3 (ISBN eBook)
978-1-4666-5867-7 (ISBN Print and Perpetual)

This book is published in the IGI Global book series Advances in Data Mining and Database Management (ADMDM) (ISSN: 2327-1981; eISSN: 2327-199X)

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 18
Accessing Big Data in the Cloud Using Mobile Devices

Haoliang Wang
George Mason University, USA

Wei Liu
University of Rochester, USA

Tolga Soyata
University of Rochester, USA

ABSTRACT

The amount of data acquired, stored, and processed annually over the Internet has exceeded the processing capabilities of modern computer systems, including supercomputers with multiple-Petaflop processing power, giving rise to the term Big Data. Continuous research efforts to implement systems to cope with this insurmountable amount of data are underway. The authors introduce the ongoing research in three different facets: 1) in the Acquisition front, they introduce a concept that has come to the forefront in the past few years: Internet-of-Things (IoT), which will be one of the major sources for Big Data generation in the following decades. The authors provide a brief survey of IoT to understand the concept and the ongoing research in this field. 2) In the Cloud Storage and Processing front, they provide a survey of techniques to efficiently store the acquired Big Data in the cloud, index it, and get it ready for processing. While IoT relates primarily to sensor nodes and thin devices, the authors study this storage and processing aspect of Big Data within the framework of Cloud Computing. 3) In the Mobile Access front, they perform a survey of existing infrastructures to access the Big Data efficiently via mobile devices. This survey also includes intermediate devices, such as a Cloudlet, to accelerate the Big Data collection from IoT and access to Big Data for applications that require response times that are close to real-time.

DOI: 10.4018/978-1-4666-5864-6.ch018

INTRODUCTION

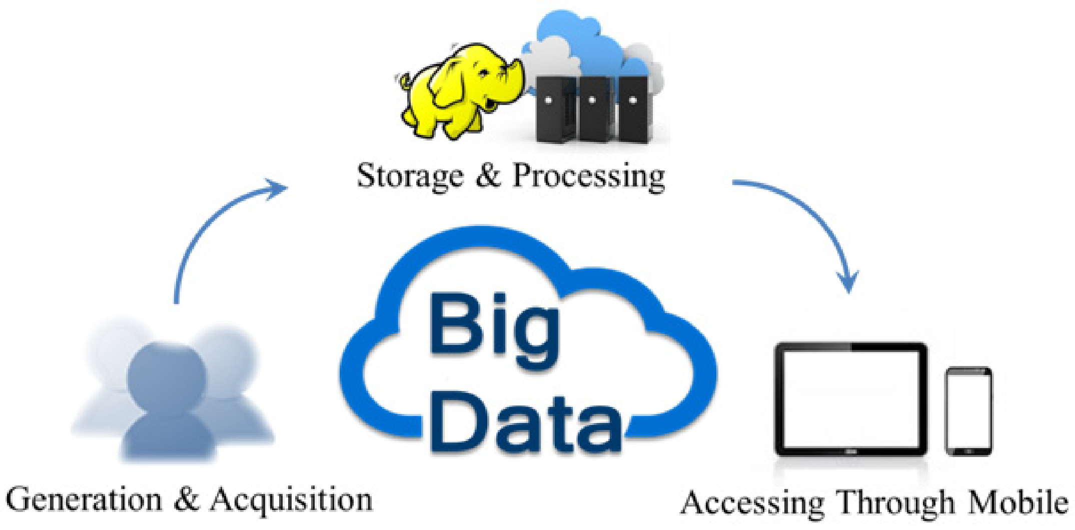
The amount of data generated annually over the Internet has exceeded the zetabyte levels. Processing data with such high volume far exceeds the computational capabilities of today's datacenters and computers, giving rise to the term *Big Data*. Although the growth rate of supercomputers that are capable of processing such explosive amount of data is also breathtaking (TOP500, n.d.), the rate of data growth far surpasses the capabilities of even the fastest supercomputers available today. Even though the top supercomputers are able to handle Big Data analysis, their highly-specialized designs are not affordable for commercial use. Instead, large commodity computer clusters are used, where faults are common and interconnect speeds are limited. Also the storage and management of Big Data poses different unique challenges: While the storage has to be performed by high-availability and high-performance distributed file systems, it must also be done in a way to allow application of efficient data analytics later. Being able to perform analytics on this data is crucial: It has been reported that, performing analytics on Big Data can save the government 14% all across their budget (Big Data, 2013). This specific example shows the importance of manipulating Big Data while keeping both phases of usage in mind concurrently: storage and computation.

By today's standards, considering the utility computing (termed *Cloud Computing*), is unavoidable for any organization, regardless of its size. While it is possible for different organizations to build their own datacenters, it is an expensive business proposition to do so, since the economies of scale for organizations such as Amazon (AWS, n.d.), Google (Google, n.d.), and Microsoft (Microsoft, n.d.), will allow them to build these datacenters for a fraction of the price. Furthermore, while an organization that is building its own datacenter must size it for the worst case, cloud operators offer much more favorable pricing options, such as, per-hour usage pricing. This allows corporations to rent much higher peak amounts of computational power with zero upfront investment. To make cloud computing even more appealing, the responsibility of continuously upgrading the underlying computational infrastructure is shifted to the cloud operators, thereby permitting access to modern high performance resources whenever they are available without any investment.

Due to the wide scope of Big Data and cloud computing, we restrict our focus to futuristic concepts involving Big Data in this chapter. Specifically, we will investigate one emerging source of Big Data, called *Internet of Things* (IoT). IoT, introduced in 1999, conceptualizes a network of numerous data-generating devices (things) such as home energy meters, wireless sensors, and other sensory devices. For IoT to be realized, a unique Internet addressing scheme for each device, called IPv6, is necessary that significantly expands what used to be the standard a decade ago (IPv4). With the widespread use of IPv6, each device (i.e., thing) can be assigned its unique address to globally identify it over the Internet. The acceptance of IPv6 is accelerating for desktop PCs and is expected to expand over to IoT within the following decade.

Cloud computing, as a new model for delivering computing resources on demand, provides a powerful, flexible and elastic platform which enables collection, analytics, processing and visualization of Big Data. Storage of Big Data is performed by file systems that are drastically different than traditional file systems such as NT File System (NTFS). One such user-level distributed file system – Google File System (GFS) allows not only the distributed storage of Big Data, but also its access with high availability (and fault-tolerance) due to the built-in redundancy in GFS. This file system also dictates how the processing should be performed: Standardized methods, such as MapReduce, ease the handling of Big Data and provide a tool for cloud operators to make their platform more accessible. Cloud computing service providers have already releases of the public

Figure 1 Illustration for the lifecycle of big data (Generation, storage and processing, accessing)



platforms for Big Data analysis (Amazon Elastic MapReduce and Google BigQuery).

Access to Big Data in the cloud through mobile devices (termed *Mobile-Cloud Computing*) significantly expands the reach of Big Data due to the widespread availability of smartphones and tablets. While multiple definitions are available in the literature (Dinh et al., 2011; Fernando et al., 2013), mobile-cloud computing can be defined as the "co-execution of a mobile application within the expanded mobile/cloud computational platforms to optimize an objective function (Soyata et al., 2013)." An objective function can be defined for the mobile application such as the *application response time*, and the goal of the mobile application is to minimize this objective function. In applications requiring real-time response (e.g., real-time face recognition), mobile devices cannot achieve this objective function alone. Mobile-cloud computing allows the mobile device to utilize cloud resources to achieve this goal.

This chapter is organized as the lifecycle of Big Data shown in Figure 1: First, we will be providing a survey of IoT as a source for Big Data generation, followed by a survey of storage and computational methodologies and algorithms for Big Data in the cloud computing environment. We will conclude our chapter with an introduction of mobile-cloud computing which allows access to the Big Data in the cloud via mobile devices.

GENERATION AND ACQUISITION

As mentioned in the previous section, a portion of future Big Data will be generated by a network of numerous data-generating devices called *Internet of Things* (IoT). The phrase IoT was first presented by Kevin Ashton at Procter & Gamble (P&G) in 1999 (Ashton, 2009). The basic idea of this concept is that, the pervasive presence of varieties of things or objects, through unique addressing schemes, ubiquitous computation and communication infrastructures, are able to interact with each other and cooperate to reach a common goal. These things or objects have their own means of gathering information. Emerging technologies, including RFID, sensor, and wireless communications enable things or objects to observe, identify, and understand the world. IoT blurs the lines between the real world and the digital world by providing awareness about situations and status of things and people in digital format, bridging the real world with the digital world.

IoT will have a profound and disruptive impact on transportation, environment, living, e-health, military and defense. This new paradigm will play a leading role in the near future. The increased autonomous decision making capabilities can be used by service technologies and enterprise systems of tomorrow: the real world awareness will be provided by the IoT. Our social interactions will be greatly enhanced with information and intelligence enabling feedback and control loops which are cumbersome, slow and fault ridden. By 2025 Internet nodes may reside in everyday things – food packages, furniture, paper documents, and more (NIC, 2008).

The development of IoT depends on dynamic technical innovation in a number of important fields. First, for object identification, a ubiquitous addressing scheme is crucial., which can be offered by Radio Frequency IDentification (RFID). Second, with emerging technologies, data can be collected and processed to perceive status changes of physical objects. Third, wireless communication technologies link the real world with the digital world, by connecting each object. Finally, advances in miniaturization and nanotechnology mean things will become more integrated, providing the strong ability to interact. Eventually a full interoperability of interconnected devices will enable adaptation and autonomous behavior while guaranteeing trust, privacy, and security (Atzori, 2010). However, many issues remain to be addressed. Both industry and academia need to be involved to formulate solutions to fulfill major technological requirements before IoT is widely applicable.

The rest of this section is organized as follows. We introduce vision and applications of IoT first, followed by a presentation of key technologies which enable IoT. We conclude this section with a cloud-centric view of IoT and the issues that must be addressed before IoT is widely applicable.

Vision and Applications

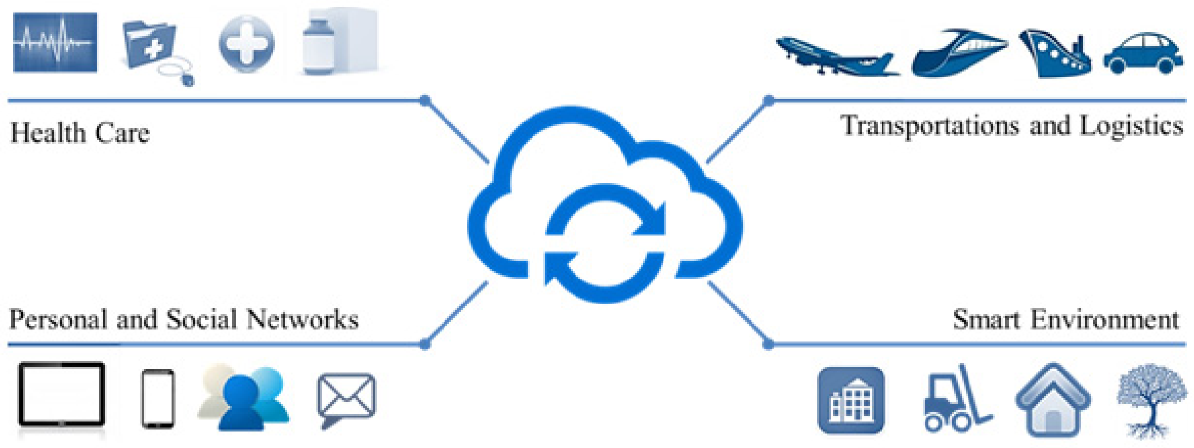
In the past 50 years, the Internet has grown from a small research network to a worldwide network with billions of human users. In the past decade, Internet of Things has evolved and became capable of connecting physical objects (smart objects). A new era of networking, computing and service provisioning and management has started (Miorandi, et al., 2012).

Conceptually, IoT is based on smart objects which are identifiable, Internet-accessible and interoperable among each other. A smart object is a physical embodiment that senses physical phenomena; and it is equipped with limited communication and computing capabilities; each smart object is associated with both a human-readable name and a unique universally identifiable machine-readable address. IoT focuses on data and information related to physical world rather than point to point communication, which distinguishes it from traditional network systems.

From a system perspective, the Internet of Things can be viewed as a highly distributed and dynamic network of many smart objects communicating with each other. Since smart objects can move and create ad hoc connections unexpectedly, the IoT network encounters a very high level of parallelism. The extremely large scale of the system makes scalability a major issue for IoT. So, self-management is expected to accelerate the development of IoT greatly (Guinard et al., 2011). From the service perspective, integration of smart objects' functionalities and resources into services (Chen et al., 2010) is a major issue, which requires a standardized representation of 'virtualized' smart objects in the information world.

The IoT has evolved as the next technology to transform the Internet to a fully integrated future Internet with a variety enabling wireless technologies like RFID tags, embedded sensor and actuator nodes. A wide range of applications can be deployed to improve the quality of our lives

Figure 2. Illustration for application areas of IoT



with IoT. Depicted in Figure 2, these applications can be itemized as follows (Atzori et al., 2010):

- **Transportation and Logistics**
  With RFID and NFC technology, real time monitoring of the entire supply chain in logistics makes it possible to obtain product-related information timely and accurately so that the customer service time can be greatly improved.
  Car drivers can benefit from the information obtained from the road system for better navigation and safety. More accurate information for planning activities can also be obtained.
- **Health Care**
  In the health care domain, real time tracking of a person or object (e.g. patient-flow monitoring) can be achieved with the IoT technology. Also, it can provide identification to prevent mismatching so that no harmful effects will occur to patients (wrong drug or time). IoT enabled data collection and sensing can help improve health care to patients as well.
- **Smart Environment**
  With sensors and actuators distributed around our living environment, IoT technology makes our living environment more comfortable in that room heating, lighting can adaptively change according to our preference and certain incidents can be avoided with appropriate monitoring and alarming. Also, with massive deployment of RFID tags, quality control can be performed to industrial plants to help improve automation quality.
- **Personal and Social**
  IoT helps people interact with each other to build social relationship by automatically and intelligently sending messages about our activities to friends. Also, lost or stolen objects can be easily identified and tracked with the attached electronic tags.

According to the IoT vision, a smart planet where the world economy and support system will seamlessly and efficiently cooperate will evolve in the future.

Enabling Technologies

Radio-Frequency Identification (RFID)

As the size, weight, energy consumption and cost of radio transmitters decrease, the possibility of integrating radio transmitters in almost anything will be the key enabler for the IoT concept. The RFID system usually consists of RFID tags embedded in every smart object and one or more readers that collect and transmit the object information (e.g. identity, location) to remote computer servers (Atzori et al., 2010). With no human interaction while monitoring the objects in real time, mapping of the real world to the virtual world becomes possible.

Physically, an RFID tag contains an IC chip for information and signal processing (RFID, n.d.) and an antenna for receiving and transmitting signals. RFID tags can be categorized into passive tags and active tags. Passive RFID tags have no power supply and can harvest energy from the electromagnetic energy received from RFID readers. Although the gain from an RFID reader is very low, tag IDs can still be correctly retrieved within a radio range of a few meters. Active RFID tags have their own power supply (e.g. a battery) on-board. The lifetime of an active tag is thus limited by the power supply. However, active tags can transmit over a much longer distance, typically a few hundred meters. RFID reader act as a gateway between physical objects with RFID tags and the Internet by resolving all the mismatches in the architecture, naming convention and communication protocols (Kopetz, 2011).

Wireless Sensor Networks

A Wireless sensor network (WSN) is an infrastructure composed of sensing, computing, and communication elements that can trace the status of things and is aware of its environment. It can act as a bridge connecting the physical world to the digital world, and can instruct administrators to react to events and phenomena in a specified way (Sohraby et al., 2007).

A WSN typically consists of density diverse sensor nodes. Each sensor node has several parts:

- A localized and application-specific sensor operating in the seismic, radio, acoustic, optical and chemical or biological domains.
- A radio transceiver with an internal or external antenna whose communication bandwidth and distance are limited.
- A micro computing unit to process signals and data.
- A battery or an embedded form of power harvest.

The sensor nodes are often aware of their locations through a local positioning algorithm or the Global Positioning System (GPS). Because of the limited communication distance, there is also another kind of node called the sink node, whose responsibility is to forward data from sensor nodes to the center node of the information cluster. Because of the small number of sink nodes, they can cost more than the sensor nodes, and therefore have a stronger communication ability.

When a sensor node is deployed in the field, it needs to self-organize a network. It first detects its neighbors and establishes communication with them. It, then, needs to learn the topology in which the nodes are connected to each other, and build an ad-hoc multi-hop communication path to a sink node. When a sensor node or a sink node fails, it must reconfigure its network.

To support the operation of nodes, it is important to have an operating system designed specifically for WSNs. Such an operating system should have a small code size which can adjust to memory constraints of nodes, and utilize modular architecture. An example is TinyOS (Levis, 2005), which is an open-source operating system designed for WSNs and low-power embedded devices. TinyOS combines flexible, fine-grain components with an execution model that supports complex yet safe concurrent operations. Its core size is about 400 Bytes.

Most commercial WSNs are based on the IEEE 802.15.4 standard. IEEE 802.15.4 specifies the fundamental physical layer and media access control for wireless personal area networks (WPANs) which focus on low-cost and low speed communication amongst devices.

Current WSNs have several limitations:

- **Power efficiency.** The lifetime of a node depends on the battery-power or harvested power and its power consumption.
- **Environment.** WSNs are often deployed in harsh environments. Nodes in WSNs may need to withstand high/low temperature, nuclear radiation, sand storm, and so on. Such environment conditions give rise to challenges in the manufacturing and management of the nodes.
- **Node cost.** There are typical hundreds and even thousands of nodes in WSNs. The cost of one node is critical to the overall cost of WSNs.

Sensing RFID systems will allow building small-size and low-power RFID sensor networks (Buettner & Wetherall, 2008), which consist of small, RFID-based sensing and computing devices, and RFID readers. Nodes in this system transmit data generated by sensing RFID tags and provide the power for network operations. Their lifetime is usually not limited by the battery duration. This technology has the potential of producing long-lasting, low-cost ubiquitous sensor nodes that may revolutionize many embedded applications.

The WISP (Wireless Identification and Sensing Platform) project from Intel Research is a sensing and computing device that is powered and read by off the shelf UHF RFID readers (WISP, n.d.). WISPs have on board microcontrollers that can sample a variety of sensing devices, creating a wirelessly-networked, and battery-less sensor device. WISPs have the capabilities of RFID tags, but also support sensing and computing. Like any passive RFID tag, WISP is powered and read by a standard off-the-shelf RFID reader, harvesting the power from the reader's emitted radio signals. WISPs have been used to sense light, temperature, acceleration, strain, liquid level, and to investigate embedded security. Integration of sensing technologies and RFID tags allow building RFID sensor network (RSN) (Guinard, 2011) which consists of RFID-based sensors, and RFID readers.

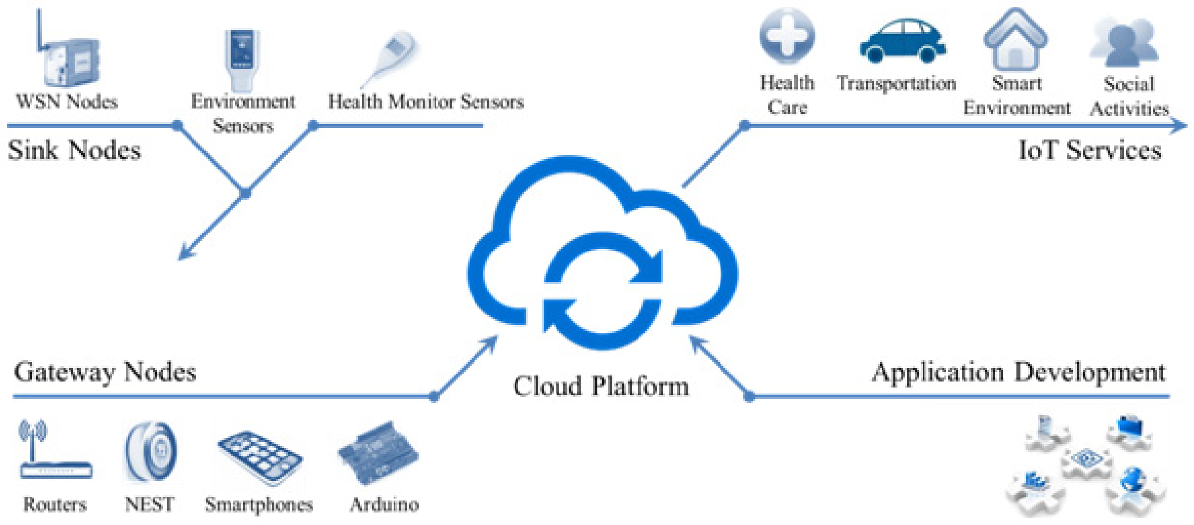
Middleware

Middleware is a software layer placed between underlying technologies and the application layer, which hides the underlying technological details and provides application interfaces, simplifying the development of new applications. Recent proposed middleware architecture often follows Service-Oriented Architecture (SOA), which is based on discrete pieces of software that provides application functionality called *service*. A *service* is a self-contained representation of reusable functions (SOA, n.d.). The purpose of SOA is to provide an easy way to cooperate large number of objects or things connected over a network. In an SOA environment, objects on the network make their resources available to others as an independent service in a standardized way (Josuttis, 2007).

An SOA solution for IoT composes of, in a top-down order, 1) application layer, 2) service composition layer, 3) service management layer, 4) object abstraction layer and 5) object layer (Atzori, 2010). Application layer provides application interfaces. The service composition layer provides independent services to build specific applications. The independent services are provided by objects in the network. Service management layer manages the objects over the network including object discovery, service deployment and status monitoring. The object abstraction layer provides standard interfaces for object access.

One challenge of SOA is managing metadata. In an SOA-based solution, it becomes complex to manage the way many services interact. Another challenge is that conventional application-man-

Figure 3. Illustration for the architecture of cloud-based sensing networks



aged security is sufficient, since the application exposes itself as a service to the outside world, which would be used by other untrusted applications.

Internet of Things and Cloud Computing

A framework for integrating ubiquitous sensing devices and the cloud provides great flexibility and scalability for IoT systems. Sensing devices can join the network and provide data to the cloud and the cloud can analyze the data and offer such infrastructure services as shown in Figure 3.

A cloud platform using Manjrasoft Aneka and Microsoft Azure (Microsoft, n.d.) utilizes a hybrid cloud (combining private and public cloud) to provide computing, storage and visualization to form a seamless framework for IoT systems (Kürschner et al., 2008). It provides a clear framework of cloud APIs for IoT applications to easily utilize Cloud services and greatly reduce development time and cost. An important feature of Aneka is that it provisions both resources on public clouds (e.g. Microsoft Azure) and resources on private clouds (e.g. clusters and virtual data centers). When scheduling an application, it determines whether to use private clouds or public clouds based on the QoS requirements of the application. The platform handles interoperability of multiple clouds by providing a standard framework for various clouds.

Open Issues

Besides the technologies that drive IoT development we discussed in the previous section, a lot more research is required to make the IoT feasible. Current issues include standardization, naming and identification, as well as security and privacy:

Standardization

Several standardizations of IoT have emerged in the scientific research communities across the globe. EPC global (Kürschner et al., 2008) enables sharing related product information by providing standardization of integrating RFID into the EPC framework (Hada & Mitsugi, 2011). GRIFS provides a standard for the transition from localized RFID to the IoT. 6LoWPAN (Kushalnagar et al., 2007) aims at making IPv6 protocol compatible with current low power IEEE 802.15.4 devices.

ROLL (Weiser, 1999) gives a definition for a routing protocol for future generation Internet networks that are heterogeneous low power. With the cooperation of the industry that provides standardizations in different areas, the IoT will become much more achievable.

Naming and Identification

With a large amount of addressable nodes emerging in the IoT era, a new effective addressing policy is required. The new IPv6 protocol is proposed for such low-power wireless communication nodes in the aforementioned 6LoWPAN study. The mechanism to map a reference to a description of a specific smart object and its associated RFID tag identifier was introduced to be performed by Object Name Servers (ONS). Additionally, the data traffic generated by IoT differs significantly from the traffic generated by the devices that are currently on the Internet, necessitating a new Quality of Service (QoS) support for the IoT.

Security and Privacy

The IoT is easily attacked since 1) its components are usually unattended, 2) its wireless communication system is easily eavesdropped and 3) the IoT components need complex security schemes. Two major problems are authentication and data integrity: In the IoT, the current authentication mechanism to exchange messages among nodes is not feasible because of limited bandwidth. Different solutions for authentication have been introduced for WSN and RFID systems, although, none of them can handle the man-in-middle attack. Passwords are usually used to ensure data integrity in the IoT, but the length of password cannot provide strong protection currently.

With the available techniques today, private personal information can be easily gathered without the knowledge of a person through IoT devices. Even if some of the proposed mechanisms are valid solutions, IoT's widespread adoption will not materialize due to such privacy concerns: Until the effectiveness of the proposed security solutions are time-tested and certain confidence levels have been established, IoT will remain in its exploratory phase. Finally, *digital forgetting* is becoming an emerging research topic in the IoT. With digital forgetting, all information will be kept forever so that any information can be retrieved using data mining techniques.

Since the mid-1990s, the Internet has had a tremendous impact on our life and society. It changed the way we interact with one another and exchange/receive information. However, the information we can access from the Internet is mainly obtained from manual-typing, taking digital pictures, or scanning. The ability to sample information from things is limited when we face the real world, because there are so many things. IoT can change the way information is sampled. The thing itself can transfer information into the network by itself, which means things become our senses (eyes, ears, and noses). IoT adds another dimension to how we access and handle information. In the past ten years, we have made a substantial progress in IoT. But the feasibility, scalability and efficiency are still limited by existing technologies, which will drive the research and development of IoT in the next decade.

STORAGE AND PROCESSING

Recently, there has been an explosive growth in the amount of data that is being generated by humans through social networks and online transactions. Alternatively, a similar growth is observed in the amount of data that is generated by machines through the sensor networks and scientific research. While all of this data may be potentially valuable, extracting the value from such massive quantities of data presents significant challenges, and was termed *Big Data*. Big Data implies datasets that are large and complex enough to the point where conventional approaches will fail to store and process them efficiently. Three dimensions have been proposed to characterize Big Data: Volume, Variety and Velocity (Laney & Beyer, 2012). Another dimension is included is the Value. These aspects of Big Data are defined below

- *Volume*: The massive quantity and high growth of data requires high horizontal scalability which outpaces conventional storage systems.
- *Variety*: The data are collected from various heterogeneous sources like social media, airplane sensor logs to DNA research projects. All these data may be analyzed altogether to generate valuable results. Conventional relational database management and analysis techniques will fail when faced with such variety.
- *Velocity*: The data are generated and collected at a high speed and the *real-time* demand for the analyzed results will require both high-performance and data-intensive processing systems.
- *Value*: Data value measures the usefulness of the Big Data for accomplishing various targets, such as, decision making. Many statistical., data mining and machine learning methods along with the data storage and processing techniques will uncover the hidden value of Big Data.

The development of cloud computing provides an on-demand cost-efficient computing platform with great horizontal scalability, which is an ideal platform for storing and processing large datasets. However, conventional techniques like relational database management systems cannot efficiently utilize the power of cloud computing.

Three major issues brought by Big Data: storage, management and analytics and their current solutions in the cloud computing configurations are discussed in the following sections.

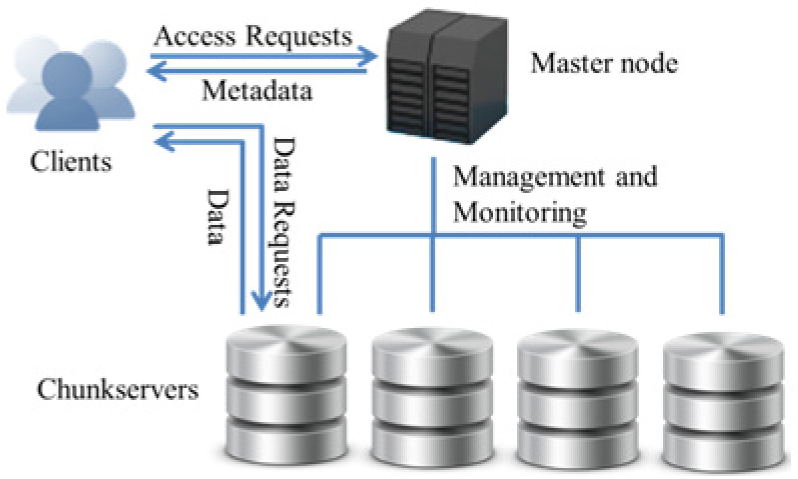
Storage

With the increasing data sizes from terabytes to petabytes to exabytes, the data can no longer be stored in a few computers. The need for distributed data storage and access within clusters, across clusters and even across datacenters brings new challenges to the existing distributed file systems. Early in 2003, Google released its own Google File System (GFS), as a scalable distributed file system for large distributed data-intensive applications (Ghemawat et al., 2003). The design of GFS is driven by three key observations in Google's environment:

- Files are very large and are growing very fast.
- File appending happens more common than overwriting.
- Component failures are the norm rather than the exceptions.

In order to provide fault tolerance on a large number of inexpensive commodity machines and deliver high aggregate performance to a large number of clients, a typical design of a GFS cluster consists of a single master node, and several chunk servers. The master node maintains all of the file system metadata including the namespace, access control information and file-to-chunk mapping, and several chunk servers contain the data that is actually stored in the form of 64MB chunks. Both master node and chunk servers are user-level processes running on Linux-based machines. As shown in Figure 4, when accessing the file, a client first communicates with the master server to obtain the metadata and then communicates with the chunk server for the actual data according to the metadata. Master node monitors the status of every chunk server and updates its metadata accordingly when a fault occurs. Data is replicated among chunkservers to enhance availability, bandwidth utilization and overall performance.

Figure 4. Illustration for the system architecture of Google File System



An open-source implementation of GFS is the Hadoop Distributed File System (HDFS) which comes from Yahoo (Shvachko, 2010a) as part of the Hadoop framework. In 2010, more than 21PB (Petabytes) of data are stored in a single HDFS cluster consisting 2000 machines hosted by Facebook (Borthakur, 2010), showing the success of this distributed file system scheme.

However, with the relentless growth of data, scalability issues still exist in both GFS and HDFS. Furthermore, with the increasing demand for interactive applications which require low latency access instead of high throughput, original designs of GFS and HDFS have significant difficulties handling I/O requests with an interactive pattern. The original GFS and HDFS designs are optimized for large files (several GBs) while Big Data doesn't necessarily consist of large files. Instead, the dataset might consist of a large number of small files which are far below the size a block (typically 64MB). Since every file, directory and the underlying block is represented as an object in the memory of the name node, based on a rule of thumb (Shvachko, 2010b), very large number of files easily saturate the memory of the name node, causing file accesses to suffer severe overhead and sometimes make it completely infeasible to access some files. One possible solution for this issue is to use a *Sequence Fil*e. The idea was introduced to bundle small files into a single sequence file and process it in a streaming fashion, which partially solves the performance problem at the expense of introducing another problem: The ability to list all files and randomly access one of them in a single sequence file is lost, leading to other projects including BigTable and HBase as an abstraction layer on top of the distributed file system to provide better performance and scalability under various situations. Another issue with the original system architecture is that, both of these systems are built upon the single-node namespace server architecture, which will naturally become the limiting point as the system scales. Distributed namespace server system was introduced by Google recently (McKusick & Quinlan, 2009) as a more promising solution to eliminate the single-name-node scalability issue. The resulting GFS system can now handle hundreds of master nodes and each file is split into much smaller chunks than before. More features including load balancing and better monitoring and recovery are also deployed in this implementation of GFS.

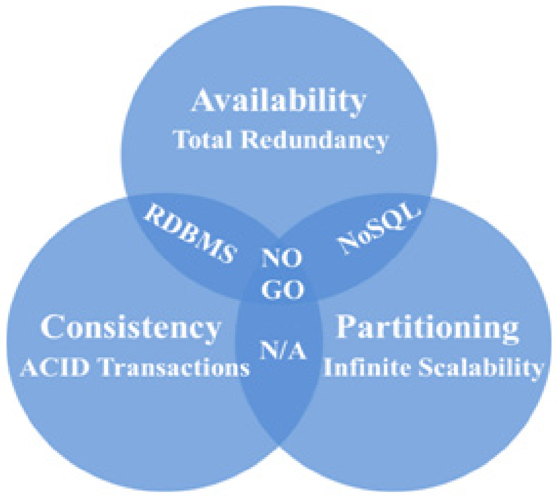
Management

Distributed file systems provide mechanisms to store massive amounts of data. However, the way this high volume (and variety) of data is efficiently organized, managed and retrieved still remains an issue for distributed database systems. Conventional relational database systems enforce integrity of complex relational data structures, thereby under-utilizing cloud computing resources and providing poor horizontal scalability. Also, as previously mentioned, heterogeneous sources generate data in various formats, ranging from structured to semi-structured or even un-structured. Most of these formats require the ability to rapidly change the underlying database structure and fit poorly with the conventional relational database systems.

The non-relational., schema-less, analytic-oriented, NoSQL databases have been growing in use, as a solution to deal with the organization and management issues of Big Data. NoSQL originally means databases that provide no support for Structured Query Language (SQL) to manipulate data while now NoSQL databases are designed to achieve better horizontal scalabilities and availabilities by compromising the consistencies and complexities of an underlying database model, as shown in Figure 5.

Based on the CAP theorem, a distributed system cannot simultaneously guarantee consistency, availability and partition tolerance. Traditional Relational Database Management Systems (RDBMS) focus on availability and consistency, providing reliable ACID (Atomicity, Consistency, Isolation and Durability) properties for transactions. However, when the system scales, it is difficult for a relational database system to be efficiently partitioned to large number of nodes, especially when the underlying data model is sophisticated. For the case of Big Data, due to its large volume, tables will grow dramatically either in size or in quantity, slowing down the query operations dramatically, especially for join operations on multiple tables. Also, an RDBMS uses fixed database schema, which is perfect for modeling conventional data. However in the case of Big Data, highly various data requires a flexible data schema or even an unknown schema that is only known by analyzing the data. This requires the proper storage data in the first place, causing a dilemma, not to mention the data that has no schema at all. Thus, the need to analyze unstructured data such as documents and log files, as well as semi-structured data such as history forms, cannot be satisfied by RDBMS.

Figure 5. Illustration of RDBMS and NoSQL within the CAP theorem



To adapt traditional database systems to the modern cloud computing architecture, conventional RDBMSs are engineered to eliminate the rule of prioritization by surrendering the strong consistency guarantees to gain significant scalability advantages. These re-engineered databases are able to fully utilize cloud computing resources. Also, to achieve higher performance and flexibility, underlying data models are greatly simplified to be schema-less. Several NoSQL databases have been developed and optimized for different data models including column-based, key-value, and document and graph. They provide much greater flexibility in representing and organizing data. The development of wide-column-based Google BigTable is aimed addressing these issues (Chang et al., 2008). BigTable appears as a sparse, distributed, persistent multidimensional sorted map (Chang et al., 2008) which provides high availability and scalability for storing structured data. Three-dimensional tables (Row, Column, and Timestamp) are optimized for GFS by being split into multiple tablets which can be accessed by special metadata tablets organized in a two-level hierarchy. Google BigTable now supports a number of Google applications and continuously evolving.

Google BigTable's open-source counterpart, HBase, released as a part of the Hadoop framework, has become one of the most popular NoSQL databases used to process and analyze Big Data (HBase, n.d.). Another popular open-source column-based NoSQL database is Apache Cassandra (Lakshman & Malik, 2010). First released by Facebook, Cassandra squashes the master-node-oriented design which makes HBase operationally inflexible. This makes Cassandra immune to single-point failures and enables it to provide higher availability and higher performance. Tunable consistency is also supported in Cassandra to provide operational flexibility.

In addition to column-oriented Google BigTable, HBase and Cassandra, there are also various NoSQL databases optimized for different data models. For example, MongoDB (Plugge E., et al., 2010) is designed for document storage while DynamoDB (DeCandia, 2007) and Voldemort (Sumbaly, 2012) are Key-Value oriented. Since the data that NoSQL databases are operating on being so divergent, there is no single universal NoSQL database that meets every requirement which necessitates the use of multiple databases in many cases. On the other hand, using multiple databases increases the cost of database maintenance. Therefore, a current trend for NoSQL database management system development is the middleware for integration of multiple hybrid back-end database engines, where various data can be automatically identified and stored in the proper database.

Although original NoSQL designers deliberately provided no consistent support, the lack of the ability to perform global ACID transactions has become one of the major drawbacks of NoSQL databases. Some early NoSQL databases provide no consistency guarantees, leaving the job to the programmers, where the conventional relational databases have significant advantages to ease program development. Early version of Google BigTable only provided single-row transactions. Some modern NoSQL designs such as DynamoDB enforce somehow stronger constraints on consistency called *Eventual Consistency*, which means that, if no new updates are made on a given item, eventually all accesses to that item will return the last updated value however any value can be returned before the system finally converges. In 2011, Megastore system with strong consistency guarantees was released by Google (Yushprakh et al., 2011), which is a schema-oriented database that supports ACID property and transactions. In 2012, Google released its globally-distributed and synchronously-replicated database system – Spanner (Corbett et al., 2012). Paxos protocol (Lamport, 2001), two-phase commit protocol and hardware-assisted time synchronization using GPS clocks and atomic clocks is used to enforce global consistency across multiple data centers. Although the achievement of global consistency for Spanner seems to be conflicting with the CAP theorem, a careful review of CAP theorem shows that the "2 of 3" formulation is misleading. Designs that require *perfec*t availability and consistency in the presence of partitions are prohibited while we can compromise the *perfect* availability to achieve a global consistent system with high availability and partition tolerances, which indicates the trend for future development of NoSQL databases.

Analytics

The value of Big Data can only be extracted by data analytics. Although many different data analytics algorithms and techniques including statistical analysis, data mining, and machine learning can be performed on Big Data, they all rely on extremely intensive computations. The way to organize the parallel and distributed computations efficiently is the key to extract the value of Big Data.

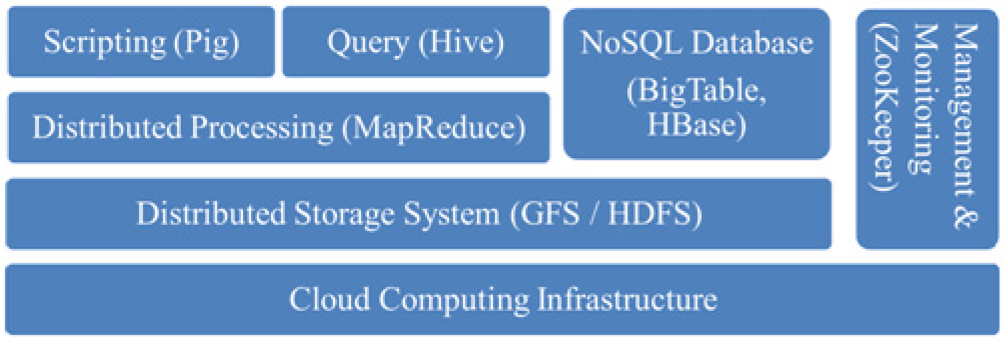
Since a large volume of data is stored in a distributed environment, traditional distributed computation paradigms and techniques like MPI, which typically bring the data to the code, will saturate the network bandwidth when feeding the data to the node before the actual computation can start, rendering the processing of large datasets infeasible. Additionally, the programmability for traditional paradigms in a massively distributed environment is significantly downgraded because of the complex computation management, coordination, synchronization, failure detection, and recovery. To address these issues, new paradigms and techniques like the MapReduce programming model are necessary. These techniques have rack-awareness in order to process the data in place and manage computation and handle faults automatically in order to simplify programming.

MapReduce paradigm was developed by Google to process large datasets stored in the distributed GFS systems (Dean & Ghemawat, 2008). Taking advantage of the distributed architecture, MapReduce pushes the computation to the node where the data resides, greatly reducing the amount of communications caused by data transfers. The computation is divided into two steps: Map and Reduce. Programmers only need to define these two functions and the framework will take care of all the rest of the entire computation, which significantly reduces the burden on the programmer and improves the robustness of the system. The open-source implementation of MapReduce model is the Hadoop framework released by Yahoo. (White, 2009)

MapReduce model and Hadoop framework are originally designed to be an offline system to support batch MapReduce applications where scalability and streaming performance are most critical. The Hadoop framework needs to be tuned to meet the real-time processing demands of OLTP (On-Line Transaction Processing) and OLAP (On-Line Analytical Processing), which have low-latency requirements, while the amount data involved in the processing is enormous. This is achieved by pipelining the Map and Reduce phases, where the Reduce phase does not wait until the Map phase finishes. The data are processed in a multiple stage pipeline. However, the system can be effectively optimized if more data is accumulated which contradicts with the low-latency requirement. To deal with this tradeoff, an adaptive flow control mechanism was introduced (Condie et al., 2010) together with incremental processing for reducers. In 2011, Facebook released its commercialized real-time Hadoop implementation to handle Facebook Messages (Borthakur et al., 2011), where HDFS and HBase are optimized for real-time transactions.

The use of input files and schema-less features of the MapReduce model prevent performance improvements available in common database systems by sing B-trees and hash partitioning (MapReduce, n.d.). This fact leads to research projects like Apache Hive and Pig for addressing some of these issues. Apache Hive is a data warehousing system used by Hadoop for querying and analysis of large data sets (Tulsa et al., 2009), where a SQL-like Hive Querying Language (HiveQL) is used to express the queries and compiled into a set of MapReduce jobs to be executed with Hadoop framework, making data manipulations much easier by squashing all of the complex and hard-to-reuse map and reduce functions. Data in the Hive is organized in a relational fashion and represented as tables, partitions and buckets which facilitate efficient data retrieval and various optimizations are built into Hive drivers and HiveQL compilers to provide better performance.

Figure 6. Building blocks for storing and processing Big Data in the cloud


Scripting (Pig)
Query (Hive)
NoSQL Database (BigTable, HBase)
Management & Monitoring (ZooKeeper)
Distributed Processing (MapReduce)
Distributed Storage System (GFS / HDFS)
Cloud Computing Infrastructure

Another similar querying project is Apache Pig. Using a similar idea to Hive, Pig provides a very simplistic scripting language called Pig Latin for data querying. The entire software stack is shown in Figure 6.

ACCESSING BIG DATA THROUGH MOBILE DEVICES

When accessing Big Data in the cloud through mobile devices, mobile-cloud computing becomes the key enabling technology in this process. With the explosion of mobile applications and the support of cloud computing for a variety of services for mobile users, mobile-cloud computing is introduced and intensively investigated as an integration of cloud computing into the mobile environment (Soyata, T., et al., 2012a; Satyanarayanan et al., 2009; Soyata et al., 2012b; Fernando et al., 2013; Cuervo et al., 2010; Chun et al., 2011; Chen et al., 2012; Verbelen et al., 2012; Soyata et al., 2013; Shi et al., 2012; Dinh et al., 2012; Kocabas et al., 2013; Guo et al., 2010; Fahad et al., 2012). Mobile-cloud computing facilities for mobile users to take full advantage of cloud computing and enables access to Big Data anywhere at any time.

In the past decade, mobile devices became increasingly more powerful to handle most of the daily operations but not powerful enough for data-intensive computations, such as querying and analyzing the Big Data. However, considering the enormous amount of mobile devices and rapid development of wireless networks, a loosely organized cluster of mobile devices can be powerful enough to collectively handle heavy computations together with the cloud, forming an integrated computing system, while maintaining the energy efficiency. To achieve such interaction and cooperation among a mobile device and multiple cloud servers, significant research has been conducted on techniques such as *Computation Offloading* and *Mobile Cloud Platform*. These techniques will be explained in the following sections.

Computation Offloading

Offloading is a solution to alleviate resource limitations on mobile devices and provide improved capabilities for these devices by migrating partial or full computations (code, status and data) to more resourceful computers (Kumar et al., 2013). The rapid development of wireless network connectivity and mobile devices in recent years has enabled the feasibility of computation offloading. Recent research efforts on computation offloading focuses on the following aspects.

- **What to offload.** The entire program cannot be offloaded for remote execution. Before offloading, the program needs to be partitioned a) manually by the programmer or b) automatically by the compiler, or c) at runtime. Manual partitioning will put the burden on the programmer, but will potentially lower the computational overhead. On the contrary, the automated partitioning can perform offloading on an unmodified program, albeit, at the expense of higher overhead. Different strategies like code tagging and dynamical prediction based on profiling can be applied to increase the performance.

- **When to offload.** Applications may have different requirements on performance and mobile devices may have different capabilities and energy concerns. Offloading decisions need to be made based on different target goals, such as a) improving performance and/or b) saving energy, or, c) reducing the network overhead. These decisions can be made by statically and/or dynamically via profiling, which has a non-negligible impact on execution overhead.
- **How to offload.** The development of virtualization and the emerging cloud computing technologies provides a powerful, flexible, manageable and secure platform for offloading, attracting significant research interest on VM (Virtual Machine)-based offloading approaches. The granularity ranges from a) OS-level to b) application/ thread-level to c) method-level.

Three computation offloading systems with different design focuses – Kimberley, CloneCloud and MAUI are briefly introduced below.

OS-Level Offloading

To achieve the goal of both high performance and manageability, the VM-based Kimberley architecture was proposed (Satyanarayanan et al., 2009). A *cloudlet*, defined as a self-managed *datacenter in a box*, was introduced in Kimberley. The cloudlet is able to support few users at a time and maintains only soft state: hence the loss of connection is acceptable.

When a mobile client connects to the cloudlet, it notifies the Kimberley Control Manager (KCM) on the cloudlet to download a small VM overlay, which is generated by comparing the target customized VM image to the base VM, from either the Internet or the mobile client. When the VM overlay is delivered, a technique called *dynamic VM synthesis* creates and launches the target VM. After the computation is done, the KCM can simply shutdown the VM and free the resources, providing self-manageability that only needs minimal maintenance.

The Kimberly system was implemented on a Nokia N810 tablet running Maemo 4.0, and the cloudlet infrastructure was implemented on a desktop computer running Ubuntu Linux where VirtualBox was used to provide the VM support. System performance was evaluated by considering the size of VM overlays and the speed of the synthesis operation. The size of generated VM overlay is around 100-200 MB for a collection of Linux applications, an order of magnitude smaller than a full VM image which can be as large as 8 GB. The processing time for VM synthesis ranged from 60 to 90 seconds and has plenty of potential room for improvements through further optimizations like parallelized compression and decompression and VM overlay prefetching.

The strengths of Kimberley are the self-manageability of the cloudlet and high flexibility for programmers to configure the code on the cloudlet since they have full control of the OS on isolated VMs. The weakness of the Kimberley design are: a) the programmer needs to decide what to offload and manually partition the program and b) the huge initialization overhead.

Thread-Level Offloading

In order to free the programmer from manual program partitioning for offloading, Chun et al. proposed the CloneCloud system, allowing the unmodified program to be accelerated by offloading a portion of the execution at the thread granularity (Chun et al., 2011). To achieve this, they modified the Dalvik VM. The modified runtime rewrites the executable of the user's program by inserting migration points via statistical analysis. When the program is running, individual threads migrate at these pre-determined migration points, from the mobile device to a device clone in the cloud, and the User Interface (UI) or other essential components continue execution on the mobile but are blocked if accessing the status of the migrated threads. A dynamic profiler is used to model the execution, migration and energy cost of each method on the mobile device, and an optimization solver is used to decide the migration points based on given optimization objectives.

An Android-based CloneCloud system prototype was implemented on an HTC G1 mobile phone and a server running the Android x86 virtual machine via VMware ESX 4.1, where the mobile clones are running. Three applications were tested on the CloneCloud prototype: a) a virus scanner, b) image search, and c) privacy preserving targeted advertising. The results show that for these tested applications, when connecting to the CloneCloud via Wi-Fi, the execution time is shortened by 2.1x-20x and the energy consumption is reduced by 1.7x-20x. When connecting to the CloneCloud via 3G, the execution time is shortened by 1.2x-16x and the energy consumption is reduced by 0.8x-14x.

The strength of the CloneCloud system is that it achieves distributed execution without manually modifying the source code, taking the *program partitioning* burden off the programmer. The weakness of CloneCloud is that, for complex applications, the overhead to transfer the state (heap and stack) may counterweigh the performance gain and energy savings of offloading. Furthermore, the security issues are not considered in the CloneCloud system.

Method-Level Offloading

Motivated by the fact that the energy consumption will remain the primary bottleneck for handheld mobile devices, MAUI (Mobile Assistance Using Infrastructure) was proposed to address this issue by minimizing energy consumption through computation offloading (Cuervo et al., 2010). Cuervo et al. observed that, the completely automated program partitioning and coarse-grained offloading will increase the overhead, thereby consuming more energy. To decrease the overhead while minimizing the burden on the programmer, they use a more fine-grained method-level offloading and the target method is identified by programmers' annotations in the source code.

MAUI is built on the Microsoft .NET Common Language Runtime (CLR) for code portability. The programmer decides which methods may be offloaded and annotates them with tags. These methods, along with the necessary program state, are extracted using reflection and type-safety. The MAUI profiler profiles each method and uses serialization to determine the offloading costs. Combining measurements of processing and transferring, a MAUI solver decides whether the method is worth offloading based on the solution to an Integer Linear Programming (ILP) formulation. MAUI generates two proxies on both the mobile device and the server that handle control and data transfer. The MAUI coordinator on the server side handles the authentications, resource allocations and executions.

The mobile part of MAUI was implemented on an HTC Fuze mobile phone running Windows Mobile 6.5 with the .NET Compact Framework v3.5, and the MAUI server was implemented on a desktop running Windows 7 with the .NET Framework v3.5. The main results measure the energy consumption and the execution time for three applications: a) face recognition, b) 400 frames of a video game, and c) 30 moves in a chess game. The results show that using remote execution on MAUI saves 5x-12x energy compared to the mobile-phone-only case. Also, MAUI reduces the execution time by more than a factor of 6.

Mobile Cloud Platform

A *Cloud* is usually considered to be a collection of powerful servers, potentially located at diverse geographical locations. However, with the increasing processing capability of mobile devices, a collection of mobile devices connected via a local ad-hoc network can now provide a powerful enough computational environment to serve as a *Mobile Cloud*. Recently, this mobile cloud concept has been investigated as a powerful and more importantly, an energy-efficient platform to support massively parallelizable applications. The potential for integrating a mobile cloud platform with the existing cloud computing architecture to form a hybrid system for Big Data has also been the focus of significant recent research. Examples of using mobile devices as a cloud of computing resources are a) Hyrax, b) NativeBOINC and c) GEMCloud and will be described below.

Hyrax

Apache Hadoop (White, 2009) is an open-source implementation of the MapReduce programming model. It is originally designed to run on powerful server clusters. To utilize mobile devices as computation units, Marinelli ported Hadoop to the Android platform and proposed the Hyrax system (Marinelli E., 2009). Hyrax enables computation jobs to be executed on distributed mobile devices connected by a wireless network.

A distributed multimedia search and sharing application were implemented on Hyrax. Experiments show that Hyrax can easily scale up to 10 HTC G1 and 5 HTC Magic mobile phones running Android 1.5 in terms of execution time and resource usage. The energy efficiency of Hyrax was shown to be significantly higher than traditional server clusters. However, the performance of Hyrax was poor compared to Hadoop on traditional servers. This is due not only to the computational capabilities and WiFi connection speed of the devices being low (ARM11 CPU @ 528MHz and 802.11g wireless router with a 54 Mbps bandwidth), but also because Hadoop was not originally designed (nor optimized) for mobile devices, causing unacceptable overhead within the system.

NativeBOINC

The NativeBOINC is an Android implementation of the BOINC (Berkeley Open Infrastructure for Network Computing) (Anderson, 2004) which is an open-source volunteer computing software utilizing crowd-sourcing for scientific computing. NativeBOINC for Android allows mobile device users to choose projects, start and stop them on demand, contributing their free computing power. Experiments show that (Eastlack, 2011) the ARM-based mobile processors have energy efficiency advantages over the traditional Intel desktop processors.

GEMCloud

GEMCloud (Green Energy Mobile Cloud) is another example of using mobile devices to create an ad hoc cloud of computing resources (Ba, 2013). By utilizing distributed mobile devices to cooperatively accomplish large parallelizable computational tasks, the author envisions that such approaches can make use of the massive amount of idle computing power that is potentially available to the public. More importantly, the authors show that a mobile computing system like GEMCloud has significant advantages in energy efficiency over traditional desktop cloud servers when the overall system is considered, rather than each individual computational device.

USING A CLOUDLET AS AN ACCELERATOR

Although mobile devices have been improved dramatically over the past few years, they are still relatively limited in processing speed, memory, storage, battery life, and network bandwidth. For latency-sensitive and compute-intensive applications, it is important to reduce the application response time to provide the best user experience. Because of the inconsistent network conditions over the Internet and the possible unavailability of cloud servers, a cloudlet can be introduced to provide local computing power and storage and the intelligence for task management (Wang, 2013; Soyata et al., 2012b; Soyata et al., 2012c). Figure 7 shows an example of a mobile-cloud architecture that utilizes a cloudlet as a local edge server that can communicate with the mobile over a local area network (LAN).

A cloudlet is able to accelerate both Big Data collection and Big Data access. As previously mentioned, Internet of Things, a major source for Big Data analytics in the near future, will provide continuous data streams from wireless sensor networks and periodical data from RFID readers. Due to the power and computational limitations of mobile devices and the large amount of data they need to transfer, an intermediate node like a cloudlet, which has a power supply, high computational capability, ample storage capability, and a direct Internet connection, is necessary for efficient data acquisition. The cloudlet collects and buffers the data from multiple sensors, organizes and preprocesses the data and sends the preprocessed data to the cloud for further analysis, reducing the energy consumption and design complexity of the sensors and improving the overall efficiency, especially under situations like unstable Internet connections and cloud server failures. For Big Data access, a cloudlet may serve as a local gateway for users, buffering, aggregating and scheduling query requests and processing and presenting the result from the cloud servers and therefore providing higher throughput, better efficiency and user experience. Here in this chapter, we will be focusing on the ways a cloudlet can help to reduce application response time and study them in detail.

We define a cloudlet as follows:

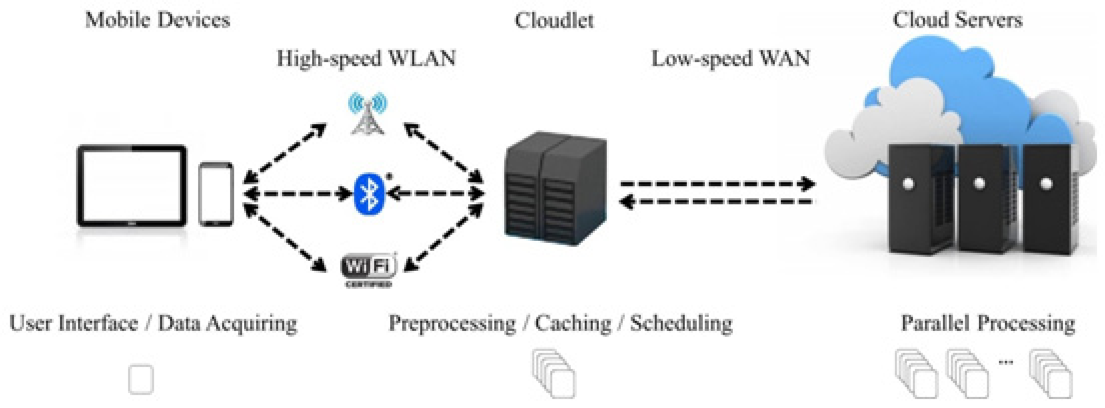
- A resourceful device which has a 10x or more performance advantage over mobile devices. It has a relatively powerful CPU and/or GPU and a large internal storage. It can support requests from tens of mobile clients and respond them fast enough, so that the responses are available to the mobile devices when they need it.
- A nearby device that communicates with mobile devices via single-hop high-speed connections such as Wi-Fi. Since a large amount of data needs to be transferred between mobile devices and the cloudlet, low-speed multi-hop connections via the WAN will counterweigh the performance gains from single-hop fast connections.
- A dedicated device for serving a target application which does not share its resources with other applications. It is equipped with a power supply, and is *always ON*. Therefore, it is capable of serving requests from mobile devices at any time.

Though similar devices have been proposed in other papers (Satyanarayanan, 2009), (Verbelen, 2012), the capability of a cloudlet to accelerate mobile-cloud computing is still unclear. Following are the three ways a cloudlet can reduce the *application response time* of a target application:

- Preprocessing.
- Caching.
- Scheduling.

These three approaches will be described in detail in the rest of this chapter.

Figure 7. Acceleration by utilizing the cloudlet as a computation and communication buffer



Preprocessing

To offload the computation to the cloud, mobile devices usually have to transmit a large amount of raw data over the Internet, which will dramatically degrade the application response time when real-time responses are desired. The cloudlet can use its higher computational capability to perform preprocessing to reduce the size of data that must be transmitted to the cloud via the Internet, thereby improving the response time. Preprocessing, from simple compression to highly sophisticated operations, can be done on the raw data to reduce its size. However, due to the limited computational power and battery life on the mobile devices, preprocessing is not a suitable candidate to perform on mobile devices. The additional computation latency may counterweigh the benefits, or even make the overall latency worse.

As shown in Figure 8, by adding a cloudlet, mobile devices now can offload the preprocessing tasks to the cloudlet via the high-speed Wi-Fi connection. The cloudlet is able to significantly accelerate the preprocessing with its powerful processor and send the preprocessed intermediate result to the cloud servers. This reduces the latency component due to the Internet data transfer time, thereby significantly improving the application response time.

Caching

The cloudlet can utilize its large internal storage to cache a portion of the big data database from the cloud so that appropriate data can be delivered over

Figure 8. Illustration of preprocessing on the cloudlet

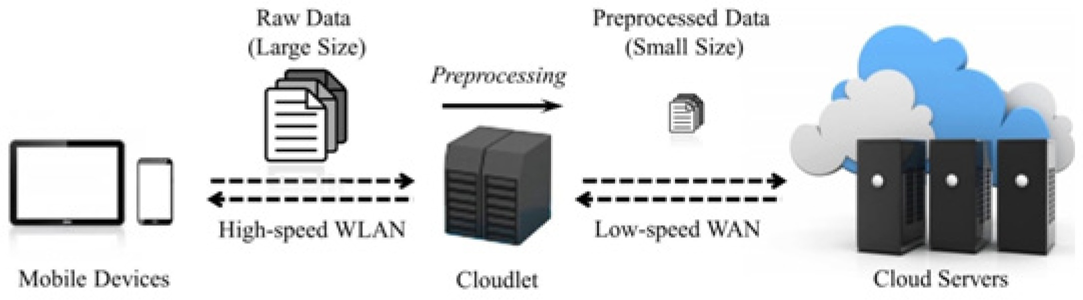
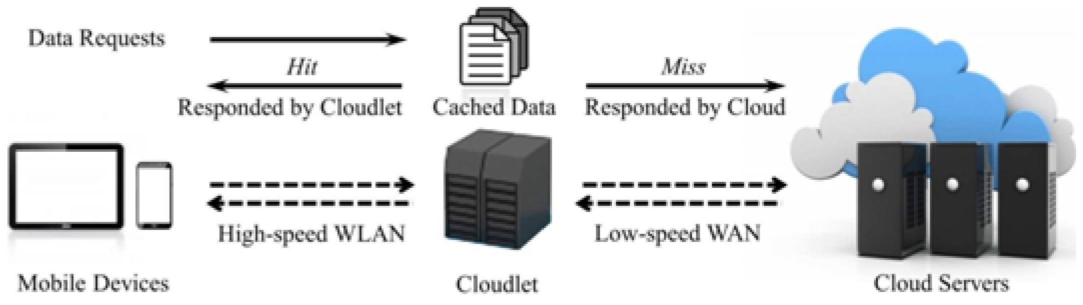


Figure 9. Illustration of caching data and computation on the cloudlet



the local network to the mobile device when the application needs this data as shown in Figure 9.

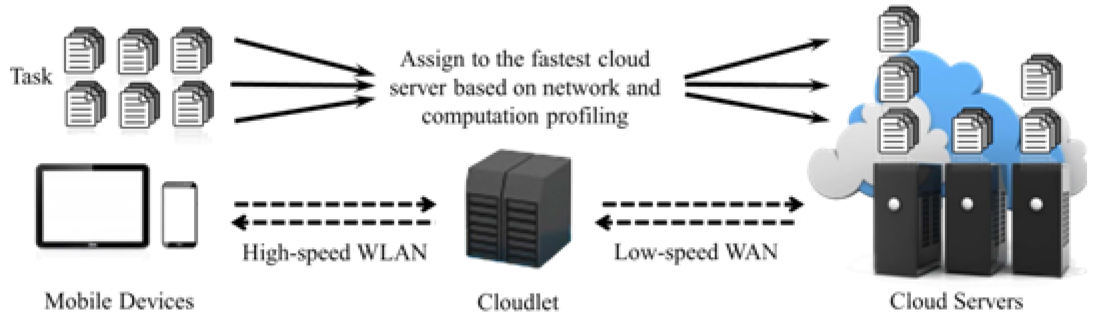
Most of the target applications are location-related and the cloudlet is designed to serve mobile devices through the local area network. Therefore, it is possible to use a cloudlet to enable fast data sharing and collaboration within nearby mobile devices. For example, if multiple mobile devices at nearby locations are performing face recognition, it is highly possible that the recognized face on one mobile device will be captured by another nearby device. Instead of sending redundant recognition requests to the cloud servers, with the presence of a cloudlet, the recognized result can be cached in the cloudlet and provided through the local network when requests from others match or hit the result. Without routing requests to the cloud, the high latency over the Internet can be eliminated and the workload on the cloud servers can be significantly filtered, providing the potential for the cloud servers to serve more mobile devices simultaneously.

Scheduling

The cloudlet has the ability to schedule multiple cloud servers and serve multiple mobile devices. The cloudlet can provide *profiling* of all the available resources to perform intelligent task distribution and optimize the overall performance to ensure a Quality of Service (QoS) goal. Sophisticated combinatorial optimization algorithms that model the cloud-to-cloudlet delays as a set of graph edges/vertices can be used for scheduling (Soyata & Friedman, 1997; Soyata & Friedman, 1999; Soyata & Friedman, 1994a; Soyata & Friedman, 1994b; Soyata et al., 1993; Soyata et al., 1995; Soyata et al., 2012c). A set of integer linear inequalities can be solved for optimum scheduling when local computational resources are available that permit the solution of computationally-intensive Integer-Linear Programming (ILP) algorithms.

In the traditional mobile-cloud computing architecture, when mobile devices offload computation to the cloud, there are usually multiple available cloud servers with different network and loading conditions. Instead of choosing a fixed server or choosing the server randomly, the mobile device should choose a server that can offer the lowest possible latency for the offloading task. The cloudlet can provide the status of each cloud to the mobile devices by continuous profiling, thereby allowing the mobile devices to choose the best possible path for computation/communication. This helps increase the battery life on mobile devices by eliminating redundant requests and network congestions when there are multiple mobile devices. Additionally, since there is no coordination between mobile devices, it is possible that several mobile devices all route their requests to one single server but leave other servers idle. This underutilization of the cloud server resources will result in poor performance and can be eliminated when a cloudlet is utilized.

Figure 10. Illustration of fairness and greedy scheduling


Task
Assign to the fastest cloud server based on network and computation profiling
High-speed WLAN
Low-speed WAN
Mobile Devices
Cloudlet
Cloud Servers

With the presence of a cloudlet, multiple mobile devices can share the profiling results provided by the cloudlet, reducing the profiling requests to the cloud servers and energy consumption on the mobile devices. Besides, as a coordinator, the cloudlet is aware of multiple mobile devices and cloud servers, based on their individual QoS requirements, several scheduling strategies can be applied by the cloudlet to increase the overall throughout and reduce the latency of a single task.

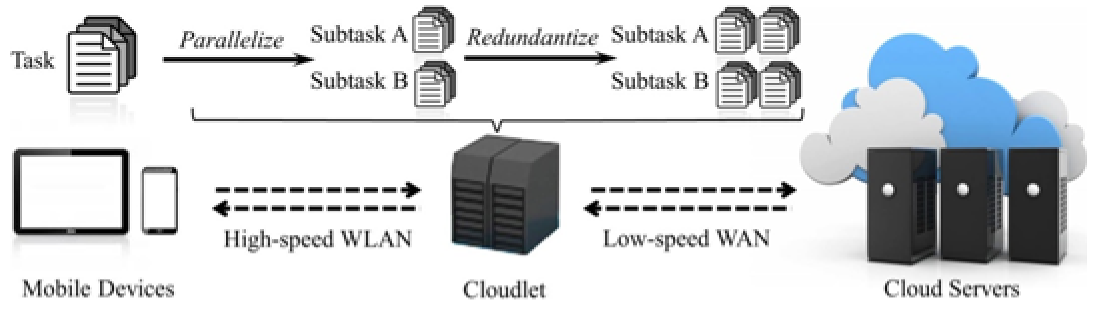
Fairness-Based Scheduling to Maximize the Throughput

For a given tolerable latency, the throughput can be maximized by scheduling tasks fairly to each cloud server according to their processing speed, current workload and network latency so that as many tasks as possible can be performed within their tolerable latency limits. A Greedy algorithm is used to decide which server should the task be scheduled for. Cloud servers with high processing speed, low current workload and high speed network connectivity should process more. In addition to this *server-side fairness*, the cloudlet can schedule the tasks in such a way that the *client-side fairness* can be achieved to avoid starvation of any one mobile decide. Prioritized scheduling can also be applied to serve important mobile clients first. The process is shown in Figure 10.

Parallelization and Redundant Scheduling to Minimize Latency

Assuming that a given task is parallelizable, the cloudlet can schedule the task to multiple cloud servers to reduce the processing and transmission latency. Considering the instability of the network and the servers' workload conditions, the gain from this parallelization may be reduced by the

Figure 11. Illustration of parallelization and redundant scheduling on the cloudlet



laggard. To alleviate the effect of such instabilities, redundant tasks can be scheduled to multiple servers (Vulimiri et al., 2012). The task is considered done when the first result comes back and other redundant tasks will be either ignored or aborted. The process is shown in Figure 11.

SUMMARY

This chapter presents a complete lifecycle for Big Data – a) generation, b) storage and processing and c) access. For the generation phase, we provided the vision of Internet of Things as a major data source in the near future. Enabling technologies including RFID, WSN and middleware are introduced together with their open issues and future directions. For the storage and processing phase, we provided an overview of the challenges brought by Big Data. State-of-the-art solutions in Big Data storage, management and analytics are introduced at a high level. Key issues for distributed systems: scalability, availability and consistency are discussed in the context of Big Data and cloud computing. For the Big Data access phase, mobile-cloud computing is described with an emphasis on computation offloading techniques and mobile cloud platforms. An intermediate node called cloudlet is proposed to accelerate the access to Big Data. Three ways that a cloudlet can help are discussed.

ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation grant CNS-1239423 and a gift from Nvidia Corporation. The authors thank Prof. Wendi Heinzelman (UR ECE), Prof. Minseok Kwon (RIT CS), Ph.D. students He Ba and Meng Zhu (UR ECE), Dr. Jiye Shi (UCB Pharma), and M.S. student Zuochao Dou (UR ECE) for their help in developing parts of the content of this chapter.

REFERENCES

Amazon. (n.d). *Amazon web services (AWS)*. Retrieved from http://aws.amazon.com

Anderson, D. P. (2004, November). Boinc: A system for public-resource computing and storage. In Proceedings of Grid Computing, (pp. 4-10). IEEE.

Ashton, K. (2009). *That 'internet of things' thing*. Retrieved from http://www.rfidjournal.com/articles/view?4986

Atzori, L., Iera, A., & Morabito, G. (2010). The internet of things: A survey. *Computer Networks*, *54*(15), 2787–2805. doi:10.1016/j.comnet.2010.05.010

Ba, H., Heinzelman, W., Janssen, C. A., & Shi, J. (2013). Mobile computing-a green computing resource. In *Proceedings of Wireless Communications and Networking Conference*. Academic Press.

Baker, J., Bond, C., Corbett, J., Furman, J. J., Khorlin, A., Larson, J., & Yushprakh, V. (2011). Megastore: Providing scalable, highly available storage for interactive services. *CIDR*, *11*, 223–234.

Big Data. (2013). *Big data analysis vs. government spending*. Retrieved from http://www.informationweek.com/government/information-management/big-data-analysis-vs-government-spending/240160233

Borthakur, D. (2010). *Facebook has the world's largest Hadoop cluster*. Retrieved from http://hadoopblog.blogspot.com/2010/05/facebook-has-worlds-largest-hadoop.html

Borthakur, D., Gray, J., Sarma, J. S., Muthukkaruppan, K., Spiegelberg, N., Kuang, H., & Aiyer, A. (2011). Apache Hadoop goes realtime at Facebook. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data* (pp. 1071-1080). ACM.

Buettner, M., Greenstein, B., Sample, A., Smith, J. R., & Wetherall, D. (2008). Revisiting smart dust with RFID sensor networks. In *Proceedings of the 7th ACM Workshop on Hot Topics in Networks* (HotNets-VII). ACM.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., & Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems*, *26*(2), 4. doi:10.1145/1365815.1365816

Chen, E., Ogata, S., & Horikawa, K. (2012). Offloading Android applications to the cloud without customizing Android. In Proceedings of Pervasive Computing and Communications Workshops (PERCOM Workshops), (pp. 788-793). IEEE.

Chen, L., Tseng, M., & Lian, X. (2010). Development of foundation models for internet of things. *Frontiers of Computer Science in China*, *4*(3), 376–385. doi:10.1007/s11704-010-0385-8

Chun, B. G., Ihm, S., Maniatis, P., Naik, M., & Patti, A. (2011). Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems* (pp. 301-314). ACM.

Condie, T., Conway, N., Alvaro, P., Hellerstein, J. M., Elmeleegy, K., & Sears, R. (2010, April). MapReduce online. *NSDI*, *10*(4), 20.

Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., & Woodford, D. (2012, October). Spanner: Google's globally-distributed database. In *Proceedings of OSDI* (Vol. 1). OSDI.

Cuervo, E., Balasubramanian, A., Cho, D. K., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. (2010). MAUI: Making smartphones last longer with code offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services* (pp. 49-62). ACM.

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107–113. doi:10.1145/1327452.1327492

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., & Vogels, W. (2007). Dynamo: Amazon's highly available key-value store. *SOSP*, *7*, 205–220. doi:10.1145/1294261.1294281

Dinh, H. T., Lee, C., Niyato, D., & Wang, P. (2011). *A survey of mobile cloud computing: Architecture, applications, and approaches*. Wireless Communications and Mobile Computing.

Douglas, L. (2012). *The importance of 'big data': A definition*. Gartner.

Eastlack, J. R. (2011). *Extending volunteer computing to mobile devices.* (Doctoral Dissertation). New Mexico State University, Albuquerque, NM.

Fahad, A., Soyata, T., Wang, T., Sharma, G., Heinzelman, W., & Shen, K. (2012). SOLARCAP: Super capacitor buffering of solar energy for self-sustainable field systems. In *Proceedings of SOC Conference* (SOCC), (pp. 236-241). IEEE.

Fernando, N., Loke, S. W., & Rahayu, W. (2013). Mobile cloud computing: A survey. *Future Generation Computer Systems*, *29*(1), 84–106. doi:10.1016/j.future.2012.05.023

Ghemawat, S., Gobioff, H., & Leung, S. T. (2003). The Google file system. *ACM SIGOPS Operating Systems Review*, *37*(5), 29–43. doi:10.1145/1165389.945450

Google. (n.d.). *Google app. engine*. Retrieved from http://code.google.com/appengine

Guinard, D., Trifa, V., Mattern, F., & Wilde, E. (2011). From the internet of things to the web of things: Resource-oriented architecture and best practices. In *Architecting the internet of things* (pp. 97–129). Berlin: Springer. doi:10.1007/978-3-642-19157-2_5

Guo, X., Ipek, E., & Soyata, T. (2010). Resistive computation: Avoiding the power wall with low-leakage, STT-MRAM based computing. [ACM.]. *ACM SIGARCH Computer Architecture News*, *38*(3), 371–382. doi:10.1145/1816038.1816012

Hada, H., & Mitsugi, J. (2011). EPC based internet of things architecture. In *Proceedings of RFID-Technologies and Applications* (RFID-TA), (pp. 527-532). IEEE.

HBase. (n.d.). *Welcome to HBase*. Retrieved from http://hbase.apache.org

Hoang, D. B., & Chen, L. (2010). Mobile cloud for assistive healthcare (MoCAsH). In *Proceedings of Services Computing Conference* (APSCC), (pp. 325-332). IEEE.

Hoang, D. T., Niyato, D., & Wang, P. (2012). Optimal admission control policy for mobile cloud computing hotspot with cloudlet. In *Proceedings of Wireless Communications and Networking Conference* (WCNC), (pp. 3145-3149). IEEE.

Josuttis, N. (2007). *SOA in practice*. Sebastopol, CA: O'Reilly.

Kocabas, O., Soyata, T., Couderc, J. P., Aktas, M., Xia, J., & Huang, M. (2013). Assessment of cloud-based health monitoring using homomorphic encryption. In *Proceedings of the 31st IEEE International Conference on Computer Design*. IEEE.

Kopetz, H. (2011). *Real-time systems: Design principles for distributed embedded applications*. Berlin: Springer. doi:10.1007/978-1-4419-8237-7

Kumar, K., Liu, J., Lu, Y. H., & Bhargava, B. (2013). A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, *18*(1), 129–140. doi:10.1007/s11036-012-0368-0

Kürschner, C., Condea, C., Kasten, O., & Thiesse, F. (2008). Discovery service design in the epcglobal network. In *Proceedings of the Internet of Things* (pp. 19-34). Berlin: Springer.

Kushalnagar, N., Montenegro, G., & Schumacher, C. (2007). *IPv6 over low-power wireless personal area networks (6LoWPANs), overview, assumptions, problem statement, and goals* (RFC4919).

Lakshman, A., & Malik, P. (2010). Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, *44*(2), 35–40. doi:10.1145/1773912.1773922

Lamport, L. (2001). Paxos made simple. *ACM Sigact News*, *32*(4), 18–25.

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., & Culler, D. (2005). TinyOS: An operating system for sensor networks. In *Ambient intelligence* (pp. 115–148). Berlin: Springer. doi:10.1007/3-540-27139-2_7

MapReduce. (n.d.). *MapReduce – Wikipedia, the free encyclopedia*. Retrieved from http://en.wikipedia.org/wiki/MapReduce

Marinelli, E. E. (2009). *Hyrax: Cloud computing on mobile devices using MapReduce (No. CMU-CS-09-164)*. Pittsburgh, PA: Carnegie-Mellon Univ.

McKusick, M. K., & Quinlan, S. (2009). GFS: Evolution on fast-forward. *ACM Queue; Tomorrow's Computing Today*, *7*(7), 10. doi:10.1145/1594204.1594206

Membrey, P., Plugge, E., & Hawkins, T. (2010). *The definitive guide to MongoDB: The noSQL database for cloud and desktop computing*. Apress.

Microsoft. (n.d.). *Windows Azure*. Retrieved from http://www.microsoft.com/windowazure

Miorandi, D., Sicari, S., De Pellegrini, F., & Chlamtac, I. (2012). Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, *10*(7), 1497–1516. doi:10.1016/j.adhoc.2012.02.016

National Intelligence Council (NIC). (2008). *Disruptive civil technologies: Six technologies with potential impacts on US interests out to 2025*. Washington, DC: NIC.

RFID. (n.d.). *Radio-frequency identification – Wikipedia, the free encyclopedia*. Retrieved from http://en.wikipedia.org/wiki/Radio-frequency_identification

Satyanarayanan, M., Bahl, P., Caceres, R., & Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing / IEEE Computer Society [and] IEEE Communications Society*, *8*(4), 14–23. doi:10.1109/MPRV.2009.82

Shi, C., Ammar, M. H., Zegura, E. W., & Naik, M. (2012). Computing in cirrus clouds: The challenge of intermittent connectivity. In *Proceedings of the First Ed. of the MCC Workshop on Mobile Cloud Computing* (pp. 23-28). ACM.

Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The hadoop distributed file system. In Proceedings of Mass Storage Systems and Technologies (MSST), (pp. 1-10). IEEE.

Shvachko, K. V. (2010). HDFS scalability: The limits to growth. *Login*, *35*(2), 6–16.

SOA. (n.d.). *Services-oriented architecture – Wikipedia, the free encyclopedia*. Retrieved from http://en.wikipedia.org/wiki/Services-oriented_architecture

Sohraby, K., Minoli, D., & Znati, T. (2007). *Wireless sensor networks: technology, protocols, and applications*. Hoboken, NJ: John Wiley & Sons. doi:10.1002/047011276X

Soyata, T. (1999). *Incorporating circuit level information into the retiming process*. (Doctoral Dissertation). University of Rochester, Rochester, NY.

Soyata, T., Ba, H., Heinzelman, W., Kwon, M., & Shi, J. (n.d.). Accelerating mobile-cloud computing. *Survey (London, England)*.

Soyata, T., & Friedman, E. G. (1994). Retiming with non-zero clock skew, variable register, and interconnect delay. In *Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design* (pp. 234-241). IEEE Computer Society Press.

Soyata, T., & Friedman, E. G. (1994). Synchronous performance and reliability improvement in pipelined ASICs. In *Proceedings of ASIC Conference and Exhibit*, (pp. 383-390). IEEE.

Soyata, T., Friedman, E. G., & Mulligan, J. H. Jr. (1993). Integration of clock skew and register delays into a retiming algorithm. In *Proceedings of Circuits and Systems* (pp. 1483–1486). IEEE. doi:10.1109/ISCAS.1993.394015

Soyata, T., Friedman, E. G., & Mulligan, J. H. Jr. (1995). Monotonicity constraints on path delays for efficient retiming with localized clock skew and variable register delay. [). IEEE.]. *Proceedings of Circuits and Systems*, *3*, 1748–1751.

Soyata, T., Friedman, E. G., & Mulligan, J. H. Jr. (1997). Incorporating interconnect, register, and clock distribution delays into the retiming process. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *16*(1), 105–120. doi:10.1109/43.559335

Soyata, T., & Liobe, J. (2012). pbCAM: Probabilistically-banked content addressable memory. In *Proceedings of SOC Conference* (SOCC), (pp. 27-32). IEEE.

Soyata, T., Muraleedharan, R., Funai, C., Kwon, M., & Heinzelman, W. (2012). Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In Proceedings of Computers and Communications (ISCC), (pp. 000059-000066). IEEE.

Soyata, T., Muraleedharan, R., Langdon, J., Funai, C., Ames, S., Kwon, M., & Heinzelman, W. (2012). COMBAT: Mobile-cloud-based compute/communications infrastructure for battlefield applications. In *Proceedings of SPIE Defense, Security, and Sensing* (pp. 84030K-84030K). International Society for Optics and Photonics.

Sumbaly, R., Kreps, J., Gao, L., Feinberg, A., Soman, C., & Shah, S. (2012). Serving large-scale batch computed data with project voldemort. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies* (pp. 18-18). USENIX Association.

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., & Murthy, R. (2009). Hive: A warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, *2*(2), 1626–1629.

TOP500. (n.d.). *TOP500 supercomputer sites*. Retrieved from http://www.top500.org/

Verbelen, T., Simoens, P., De Turck, F., & Dhoedt, B. (2012). Cloudlets: Bringing the cloud to the mobile user. In *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services* (pp. 29-36). ACM.

Vulimiri, A., Michel, O., Godfrey, P., & Shenker, S. (2012). More is less: Reducing latency via redundancy. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks* (pp. 13-18). ACM.

Wang, H. (2013). *Accelerating mobile-cloud computing using a cloudlet*. (Master Thesis). University of Rochester, Rochester, NY.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, *265*(3), 94–104. doi:10.1038/scientificamerican0991-94

White, T. (2012). *Hadoop: The definitive guide*. Sebastopol, CA: O'Reilly.

WISP. (n.d.). *WISP wiki*. Retrieved from https://wisp.wikispaces.com/

KEY TERMS AND DEFINITIONS

Cloudlet: The intermediate device between mobile devices and cloud to accelerate mobile-cloud computing.

Hadoop: An open-source Java implementation of Google's MapReduce model that supports big data applications in the cloud.

Internet of Things: The pervasive varieties of objects that can interact with each other and cooperate to reach a common goal over the Internet by using globally unique Internet addresses.

MapReduce: A programming model consisting of two logical steps—Map and Reduce—for processing massively parallelizable problems across extremely large datasets using a large cluster of commodity computers.

Mobile Application: A software application designed to run on mobile devices (e.g., smartphone, tablet).

Mobile-Cloud Computing: Executing a mobile application using the cloud resources to achieve a higher performance metric than what can be achieved with mobile computing alone (e.g., application response time).

Processing Power: Data manipulation speed of a computational platform (e.g., in TFLOPS—Tera Floating Point Operations Per Second).