HADOOP: WORKING PRINCIPLE AND APPLICATION

Surajit Mohanty*, Sameer Kumar Das, Shekharesh Barik

***Corresponding author: mohanty.surajit@gmail.com**

Abstract: Hadoop is a main buzz phrase and new curve for IT today. Big data is driven data with high velocity, volume, variety, veracity and value. It comes from different sources like mobile devices, internet, social media, sensors, geospatial devices and other machine-generated data. Traditional data processing and analysis of structured data using RDBMS and data warehousing no longer satisfy the challenges of Big Data. Due to the high velocity and volume of big data, the effective option is to store the big data through Hadoop, because it has capability to store and process massive amount of big data. Hadoop offers the big data implementation in small and medium sized businesses. This paper presents the working with Hadoop and its implementation in various sectors that include healthcare, networking security, market and business, sports, education system, gaming and telecommunications.

Keywords: Big data, Hdfs, Streaming, MapReduce, Combiner

1. Introduction

Big Data is a collection of large or complex data sets that cannot be processed using traditional computing techniques. Challenges include analysis, capture, data curation, search, sharing, storage, transfer, visualization and information privacy. It is not a single technique or a tool, rather it involves many areas of business and technology. We live in the data age. It's not easy to measure the total volume of data stored electronically, but an IDC estimate put the size of the "digital universe" at 6 zetta bytes in 2014 and is forecasting a tenfold growth by 2020 to 44zetta bytes. A zetta byte is one billion terabytes. That's more than one disk drive for every person in the world. This flood of data is coming from many sources. Consider the following: The New York Stock Exchange generates about 4−5 terabytes of data per day. Facebook hosts more than 240 billion photos, growing at 7 petabytes per month.

2. History

Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library. Hadoop has its origins in Apache Nutch, an open source web search engine, itself a part of the Lucene project. Nutch was started in 2002, and a working crawler and search system quickly emerged. However, its creators realized that their architecture wouldn't scale to the billions of pages on the Web. Help was at hand with the publication of a paper in 2003 that described the architecture of Google's distributed file system, called GFS, which was being used in production at Google. In 2004, Google published the paper that introduced MapReduce to the world. Early in 2005, the Nutch developers had a working MapReduce implementation in Nutch, and by the middle of that year all the major Nutch algorithms had been ported to run using MapReduce and NDFS. In February 2006 they moved out of Nutch to form an independent subproject of Lucene called Hadoop. At around the same time, Doug Cutting joined Yahoo!, which provided a dedicated team and the resources to turn Hadoop into a system that ran at web scale. This was demonstrated in February 2008 when Yahoo! announced that its production search index was being generated by a 10,000-core Hadoop cluster. Cutting named it after his son's toy elephant. It was originally developed to support distribution for the Nutchsearch engine project.

3. Working Principles Of Hadoop

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models [2]. The Hadoop framework application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

Hadoop consists of the Hadoop Common package, which provides file system and OS level abstractions, a MapReduce engine (either MapReduce/MR1 or YARN/MR2)and the Hadoop Distributed File System (HDFS). The Hadoop Common package contains the necessary Java ARchive (JAR) files and scripts needed to start Hadoop. The package also provides source code, documentation, and a contribution section that includes projects from the Hadoop Community. For effective scheduling of work, every Hadoop-compatible file system should provide location awareness: the name of the rack (more precisely, of the network switch) where a worker node is. Hadoop applications can use this information to run work on the node where the data is, and, failing that, on the same rack/switch, reducing backbone traffic. HDFS uses this method when replicating data to try to keep different copies of the data on different racks. The goal is to reduce the impact of a rack power outage or switch failure, so that even if these events occur, the data may still be readable. A small Hadoop cluster includes a single master and multiple worker nodes. The master node consists of a job tracker and name node. A slave or worker node acts as both a data node and task tracker, though it is possible to have data-only worker nodes and compute-only worker nodes. These are normally used only in nonstandard applications. Hadooprequires Java Runtime Environment (JRE) 1.6 or higher. The standard startup and shutdown scripts require that Secure Shell (ssh) be set up between nodes in the cluster.[19]In a larger cluster, the HDFS is managed through a dedicated name node server to host the file system index, and a secondary name node that can generate snapshots of the name node's memory structures, thus preventing file-system corruption and reducing loss of data. Similarly, a standalone job tracker server can manage job scheduling. In clusters where the Hadoop MapReduce engine is deployed against an alternate file system, the name node, secondary name node, and data node architecture of HDFS are replaced by the file-system-specific equivalents. At its core, Hadoop has two major layers namely: Storage layer (Hadoop Distributed File System), and Processing/Computation layer (MapReduce).

3.1 Hadoop Distributed File System (HDFS)

HDFS is a file system designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware [1]. The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. It is highly fault-tolerant and is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications having large datasets.

Streaming data access: HDFS is built around the idea that the most efficient data processing pattern is a write once,read-many-times pattern. A dataset is typically generated or copied from source,and then various analyses are performed on that dataset over time. Each analysis will involve a large proportion, if not all, of the dataset, so the time to read the whole dataset is more important than the latency in reading the first record.

Commodity hardware: Hadoop doesn’t require expensive, highly reliable hardware. It’s designed to run on clusters of commodity hardware (commonly available hardware that can be obtained from multiple vendors) for which the chance of node failure across the cluster is high, at least for large clusters HDFS has the concept of a block and it is a much larger unit i.e multiple of 64 MB or 128 MB by default. But unlike a file system for a single disk, a file in HDFS that is smaller than a single block does not occupy a full block’s worth of underlying storage. (For example, a 1 MB file stored with a block size of 128 MB uses 1 MB of disk space, not 128 MB.) Blocks fit well with replication for providing fault tolerance and availability. To insure against corrupted blocks and disk and machine failure, each block is replicated to a small number of physically separate machines (typically three). If a block becomes unavailable, a copy can be read from another location in a way that is transparent to the client.

An HDFS cluster has two types of nodes operating in a master−worker pattern: a *Name Node* (the master) and a number of *Data Nodes* (slave). The name node manages the file system namespace. It maintains the file system tree and the metadata for all the files and directories in the tree. This information is stored persistently on the local disk in the form of two files: the namespace image and the edit log. The name node also knows the data nodes on which all the blocks for a given file are located; however, it does not store block locations persistently, because this information is reconstructed from data nodes when the system starts. A *client* accesses the file system on behalf of the user by communicating with the name node and data nodes. data nodes are the workhorses of the file system. They store and retrieve blocks when they are told to (by clients or the name node), and they report back to the name node periodically with lists of blocks that they are storing. Without the name node, the file system cannot be used. In fact, if the machine running the name node was obliterated, all the files on the file system would be lost since there would be no way of knowing how to reconstruct the files from the blocks on the data nodes. For this reason, it is important to make the name node resilient to failure, and Hadoop provides two mechanisms for this.

The first way is to back up the files that make up the persistent state of the file system metadata. Hadoop can be configured so that the name node writes its persistent state to multiple file systems. These writes are synchronous and atomic.

It is also possible to run a *secondary name node*, which despite its name does not act as a name node. Its main role is to periodically merge the namespace image with the edit log to prevent the edit log from becoming too large. The secondary name node usually runs on a separate physical machine because it requires plenty of CPU and as much memory as the name node to perform the merge. It keeps a copy of the merged namespace image, which can be used in the event of the name node failing. However, the state of the secondary name node lags that of the primary, so in the event of total failure of the primary, data loss is almost certain. The usual course of action in this case is to copy the name node's metadata files that are on NFS to the secondary and run it as the new primary.

Block Caching: Normally a data node reads blocks from disk, but for frequently accessed files the blocks may be explicitly cached in the data node's memory, in an off-heap *block cache*. By default, a block is cached in only one data node's memory, although the number is configurable on a per-file basis. Job schedulers (for MapReduce, Spark, and other frameworks) can take advantage of cached blocks by running tasks on the data node where a block is cached, for increased read performance.

HDFS Federation: The name node keeps a reference to every file and block in the file system in memory, which means that on very large clusters with many files, memory becomes the limiting factor for scaling. HDFS federation, introduced in the 2.x release series, allows a cluster to scale by adding name nodes, each of which manages a portion of the file system namespace. Under federation, each name node manages a *namespace volume*, which is made up of the metadata for the namespace, and a *block pool* containing all the blocks for the files in the namespace. Namespace volumes are independent of each other, which means name nodes do not communicate with one another, and furthermore the failure of one name node does not affect the availability of the namespaces managed by other name nodes. Block pool storage is not partitioned, however, so data nodes register with each name node in the cluster and store blocks from multiple block pools. To access a federated HDFS cluster, clients use client-side mount tables to map file paths to name nodes.

HDFS High Availability: The combination of replicating name node metadata on multiple file systems and using the secondary name node to create checkpoints protects against data loss, but it does not provide high availability of the file system. The name node is still a single point of failure (SPOF). If it did fail, all clients including MapReduce jobs would be unable to read, write, or list files, because the name node is the sole repository of the metadata and the file-to-block mapping. In such an event, the whole Hadoop system would effectively be out of service until a new name node could be brought online. To recover from a failed name node in this situation, an administrator starts a new primary name node with one of the file system metadata replicas and configures data nodes and clients to use this new name node. The new name node is not able to serve requests until it has (i) loaded its namespace image into memory, (ii) replayed its edit log, and (iii) received enough block reports from the data nodes to leave safe mode. On large clusters with many files and blocks, the time it takes for a name node to start from cold can be 30 minutes or more. The long recovery time is a problem for routine maintenance, too. In fact, because unexpected failure of the name node is so rare, the case for planned downtime is actually more important in practice.

Hadoop 2 remedied this situation by adding support for HDFS high availability (HA). In this implementation, there are a pair of name nodes in an active-standby configuration. In the event of the failure of the active name node, the standby takes over its duties to continue servicing client requests without a significant interruption. A few architectural changes are needed to allow this to happen: The name nodes must use highly available shared storage to share the edit log. When a standby name node comes up, it reads up to the end of the shared edit log to synchronize its state with the active name node, and then continues to read new entries as they are written by the active name node. Data nodes must send block reports to both name nodes because the block mappings are stored in a name node's memory, and not on disk. Clients must be configured to handle name node failover, using a mechanism that is transparent to users. The secondary name node's role is subsumed by the standby, which takes periodic checkpoints of the active name node's namespace.

To understand the HDFS theory let's take an example. We have a file(file.txt) of size 200 MB. Manually we have taken a block of size 64 MB so the file.txt will be further sub divided into four sub files as follows:

a.txt(64 MB),
b.txt(64 MB),
c.txt(64 MB) and
d.txt(8 MB).

The advantage of HDFS is that the fourth block which is also of 64 MB but we are using only 8 MB of space so the rest free memory will not be wasted as in general file system rather it will be used for further work. From the fig the client will send a request to name node that I want to store file.txt in your cluster. Now name node will take care of the request and create a meta data by taking the input about the file name, it's size and the blocks i.e the number of input split(a.txt, b.txt, c.txt and d.txt) and provide the data nodes number(1,3,5,7) as a response to request. Now the client is approaching datanode1 and store the a.txt file and so on. As the system is made up of commodity hardware so there is a chance of system failure and in case datanode1 becomes dead then the retrieval of data will not be possible and it will result in data lost when the client will require it. To overcome this problem HDFS has provided a solution, by default it will keep three replication of data. In this case node1 data is copied to data node 3 and 5 and aack will be given back to data node1 that a.txt is stored and finally the data node1 will give the ack to the client that your file (a.txt) has been stored in data node.

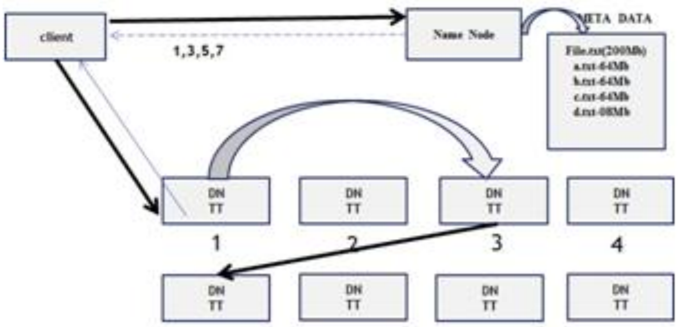

Fig.1 Storage process

3.2 MapReduce

Map Reduce is a parallel programming model for writing distributed applications devised at Google for efficient processing of large amounts of data (multi-terabyte data-sets), on large clusters (thousands of nodes) of commodity. A Map Reduce *job* is a unit of work that the client wants to be performed: it consists of the input data, the Map Reduce program, and configuration information. Hadoop runs the job by dividing it into *tasks*, of which there are two types: map tasks and reduce tasks. The tasks are scheduled using YARN and run on nodes in the cluster. If a task fails, it will be automatically rescheduled to run on a different node.

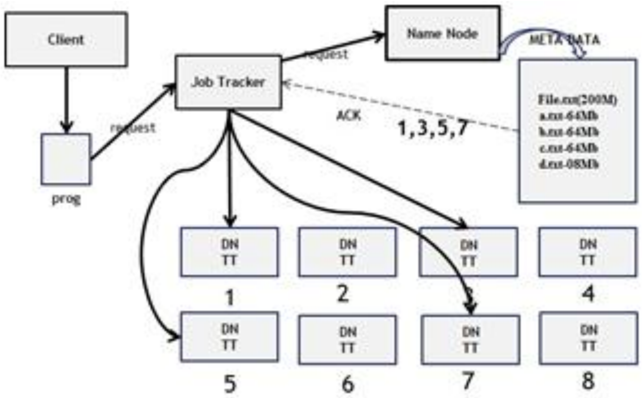

Fig.2 Retrieval of data

Now we will continue the HDFS example to see how mapping is done.

Client will send a job request in a form of programme (may be in java or in any scripting language) to job tracker which in turn will send a request to name node and ask where file.txt is?. The name node will check in file.txt in meta data and if it get the availability of the file.txt then it will reply the ack with meta data of file.txt. From that meta data it will get the data node where the files (block) is. Now the job tracker will send a request to task tracker, task tracker will check with the data node and send aack back to job tracker. This process of sending request to data node through the task tracker and getting the +ve ack about the data is called mapping.

Hadoop divides the input to a MapReduce job into fixed-size pieces called input splits, or just splits. Hadoop creates one map task for each split, which runs the user-defined map function for each record in the split. Having many splits means the time taken to process each split is small compared to the time to process the whole input. So if we are processing the splits in parallel, the processing is better load balanced when the splits are small, since a faster machine will be able to process proportionally more splits over the course of the job than a slower machine. Even if the machines are identical, failed processes or other jobs running concurrently make load balancing desirable, and the quality of the load balancing increases as the splits become more fine grained. On the other hand, if splits are too small, the overhead of managing the splits and map task creation begins to dominate the total job execution time. For most jobs, a good split size tends to be the size of an HDFS block, which is 128 MB by default, although this can be changed. Hadoop does its best to run the map task on a node where the input data resides in HDFS, because it doesn't use valuable cluster bandwidth. This is called the data locality optimization. Sometimes, however, all the nodes hosting the HDFS block replicas for a map task's input split are running other map tasks, so the job scheduler will look for a free map slot on a node in the same rack as one of the blocks. Very occasionally even this is not possible, so an off-rack node is used, which results in an inter-rack network transfer. It should now be clear why the optimal split size is the same as the block size: it is the largest size of input that can be guaranteed to be stored on a single node. If the split spanned two blocks, it would be unlikely that any HDFS node stored both blocks, so some of the split would have to be transferred across the network to the node running the map task, which is clearly less efficient than running the whole map task using local data. Map tasks write their output to the local disk, not to HDFS. Why is this? Map output is intermediate output: it's processed by reduce tasks to produce the final output, and once the job is complete, the map output can be thrown away. So, storing it in HDFS with replication would be overkill. If the node running the map task fails before the map output has been consumed by the reduce task, then Hadoop will automatically rerun the map task on another node to re-create the map output.

Combiner Functions: Many MapReduce jobs are limited by the bandwidth available on the cluster, so it pays to minimize the data transferred between map and reduce tasks. Hadoop allows the user to specify a *combiner function* to be run on the map output, and the combiner function's output forms the input to the reduce function. Because the combiner function is an optimization, Hadoop does not provide a guarantee of how many times it will call it for a particular map output record, if at all. In other words, calling the combiner function zero, one, or many times should produce the same output from the reducer.

4. Application of Hadoop

Big data applications solve and analyze real world problems using Hadoop and associated tools. Internet users and machine-to-machine connections are causing the data growth. Real time areas are defined following in which big data is used:

4.1 Healthcare

Healthcare practices and policies differ tremendously around the world, there are three objectives regarding healthcare system [4]. The first objective is to improve the patient experience (including quality and satisfaction). Second, improving overall population health and reducing the cost of health care and third is traditional methods have fallen short to manage healthcare and create modern technology to analyze large quantities of information. It is time consuming for clinical staff to Collecting massive amounts of data in healthcare. High-performance analytics are new technologies making easier to turn massive amounts of data into relevant and critical insights used to provide better care.

4.2 Network Security

Big data is changing the landscape of security technologies. The tremendous role of big data can be seen in network monitoring, forensics and SIEM [5]. Big data can also create a world where maintaining control over the revelation of our personal information is challenged constantly. Present analytical techniques don't work well at large scales and end up producing false positives that their efficacy is undermined and enterprises move to cloud architectures and gather much more data, the problem is becoming worse. Big data analytics is an effective solution for processing of large scale information as security is major concern in enterprises. Fraud detection is uses for big data analytics. Phone and credit card companies have conducted large-scale fraud detection for decades. Mainly big data tools are particularly suited to become fundamental for forensics and ATP.

4.3 Market And Business

Big Data is the biggest game-changing opportunity for sales and marketing, since 20 years ago the Internet went main stream, because of the unprecedented array of insights into customer needs and behaviours it makes possible [3]. But many executives who agree that this is true aren't sure how to make the most of it and they also find themselves faced with overwhelming amounts of data and rapidly changing customer behaviours, organizational complexity and increased competitive pressures. According to Gartner, 50% internet connection between Internet of things (IoT) devices and number reached over 15 billion in 2011 and 30 billion by 2020[18]. Some companies are succeeding at turning that Big Data promise into reality. Those that use Big Data and analytics effectively show profitability and productivity rates that are 5–6% higher than those of their peers. The companies that succeed aren't the ones who have the most data, but the ones who use it best.

4.4 Sports

Sport, in business, an increasing volume of information is being collected and captured. Technological advances will fuel exponential growth in this area for the foreseeable future, as athletes are continuously monitored by tools as diverse as sports daily saliva, GPS systems and heart rate monitors tests. These statistics and many more like them are high performance in Big Data. These numbers there is a massive amount of potential insight and intelligence for trainers, administrators, coaches, athletes, sports medics and players. Statistics can be analyzed and collected to better understand what are the critical factors for optimum performance and success, in all facets of elite sport. Injury prevention, competition, Preparation, and rehabilitation can all benefit by applying this approach. Recruitment, Scouting and retention can also be enhanced by these powerful principles. Keeping an eye on various information a coach or a manager can easily and quickly understand which athletesandplayers need additional support, training, and guidance. Areas for reasons for success and improvement will be understood more clearly. Used consistently this is a powerful measure of progress and performance.

4.5 Education Systems

By using big data analytics in field of education systems, remarkable results can be seen [6]. Data on students online behaviour can provide educators with important insights, such as if a student requires more attention, the class understanding of a topic is not clear, or if the course has to be modified. Students are required to answer accompanying questions as they go through the set of online content before class. By tracking the number of students that have completed the online module, the time taken and accuracy of their answers, a lecturer can be better informed of the profile of his students and modify the lesson plan accordingly. The analysis of data also clarify about the interest of student looking at time spent in online textbook, online lectures, notes etc. As result instructor can guide choosing the future path effectively.

5. Pros And Cons

5.1 Pros

1. Hadoop framework allows the user to quickly write and test distributed systems. It is efficient, and it automatic distributes the data and work across the machines and in turn, utilizes the underlying parallelism of the CPU cores.
2. Hadoop does not rely on hardware to provide fault-tolerance and high availability (FTHA), rather Hadoop library itself has been designed to detect and handle failures at the application layer.
3. Servers can be added or removed from the cluster dynamically and Hadoop continues to operate without interruption.
4. Another big advantage of Hadoop is that apart from being open source, it is compatible on all the platforms since it is Java based.
5. The cost savings are staggering: instead of costing thousands to tens of thousands of pounds per terabyte, Hadoop offers computing and storage capabilities for hundreds of pounds per terabyte as it uses commodity hardware.

5.2 Cons

Low-latency data access: Applications that require low-latency access to data, in the tens of milliseconds range, will not work well with HDFS. Remember, HDFS is optimized for delivering a high throughput of data, and this may be at the expense of latency. HBase iscurrently a better choice for low-latency access.
Lots of small files: Because the name node holds file system metadata in memory, the limit to the number of files in a file system is governed by the amount of memory on the name node. As a rule of thumb, each file, directory, and block takes about 150 bytes. So, for example, if you had one million files, each taking one block, you would need at least 300 MB of memory. Although storing millions of files is feasible, billions is beyond the capability of current hardware.

Multiple writers, arbitrary file modifications:
Files in HDFS may be written to by a single writer. Writes are always made at the end of the file, in append-only fashion. There is no support for multiple writers or for modifications at arbitrary offsets in the file. (These might be supported in the future, but they are likely to be relatively inefficient.)

6. Future Scope

Microsoft Research's *MyLifeBits* project gives a glimpse of the archiving of personal information that may become common in the near future. MyLifeBits was an experiment where an individual's interactions like phone calls, emails, documents were captured electronically and stored for later access. The data gathered included a photo taken every minute, which resulted in an overall data volume of 1 gigabyte per month. When storage costs come down enough to make it feasible to store continuous audio and video, the data volume for a future MyLifeBits service will be many times that.

Machine logs, RFID readers, sensor networks, vehicle GPS traces, retail transactions etc. are generating vast amount of data in structured and unstructured form. Big data is able to process and store that data and probably in more amounts in near future. Hopefully, Hadoop will get better. New technologies and tools that have ability to record, monitor measure and combine all kinds of data around us, are going to be introduced soon. We will need new technologies and tools for anonymzing data, analysis, tracking and auditing information, sharing and managing, our own personal data in future. So many aspects of life health, education, telecommunication, marketing, sports and business etc. that manages big data world need to be polished in future.

7. Conclusions

The ability to analyze and store massive amount of structured, unstructured and semi-structure data promises ongoing opportunities for academic institutes, businesses and government organizations. However, a common horizontal big data analytics platform is necessary to support these varieties of real time applications that include healthcare, security, market and business, sports, education system, gaming industry, telecommunications and probably many others in future. The applications have been discussed in this paper. Furthermore, challenges of big data, 5 V's volume, velocity, variety, value, veracity and cloud enabled big data with models and types are also described in this paper. The main goal of our paper is to make a survey of various big data applications that are use in IT industries or organisation to store massive amount of data using technologies (Hadoop, HIVE, NoSQL, Mapreduce and HPCC).

References

[1] "Tom White",The Definitive Guide 3rd Edition", OReilly, 56- 68, May.2012.
[2] "Hdfs architecture and its advantages" 14-15, http://www.tutorialspoint.com/hadoop/index.htm
[3] Sabia, SheetalKalra "Applications of big Data: Current Status and Future Scope" 5IEECSE31Aug, Chandigarh http://www.irdindia.co.in/journal/journal_ijacte/pdf/vol3.pdf
[4] "Big Data is the Future of Healthcare Bill Hamilton" cognizant 20-20 insights 2012
[5] Sam curry "big data fuels intelligence –driven security" RSA Security brief 2013
[6] Anthony G. Picciano, "The Evolution of Big Data and Learning Analytics in American Higher Education" Journal of Asynchronous Learning Network

Biographical Notes

Mr. Surajit Mohanty, Assistant professor in Dept. of CSE, DRIEMS, Tangi, Cuttack. He has 11 year industry and teaching experience. He completed his M-tech in the year 2010.His area of interest is in SAP, ERP, Data mining and BIG data analysis. He has published 12 paper in national and international journal.

Mr. Sameer Kumar Das working as an Asst. professor in CSE at GATE, Berhampur completed his M.Tech in computer science and engineering 2011.
His area of interest is in computer networking and BIG data analysis.

Mr. Shekharesh Barik, Assistant professor in the Dept. of CSE, DRIEMS, Tangi, Cuttack. He has 12 years of industry and teaching experience. He completed his M-Tech in the year 2010. His area of interest is in image processing, Algorithm Analysis and Design, Data mining. He has published 4 papers in national and international journals.