Database Management Systems: A NoSQL Analysis

Innocent Mapanga, Prudence Kadebu

Abstract- **Addressing today's ever increasing changes in data management needs require solutions that can achieve unlimited scalability, high availability and massive parallelism while ensuring high performance levels. The new breed of applications like business intelligence, enterprise analytics, Customer Relationship Management, document processing, Social Networks, Web 2.0 and Cloud Computing require horizontal scaling of thousands of nodes as demanded when handling huge collections of structured and unstructured data sets that traditional RDBMS fail to manage. The rate with which data is being generated through interactive applications by large numbers of concurrent users in distributed processing involving very large number of servers and handling Big Data applications has outpaced the capabilities of relational databases thereby driving focus towards the NoSQL database Adoption. NoSQL database systems have addressed scaling and performance challenges inherent in traditional RDBMS by exploiting partitions, relaxing heavy strict consistency protocols and by way of distributed systems that can span data centres while handling failure scenarios without a hitch. In this paper different database management systems are discussed and their underlying design principles namely ACID, CAP and BASE theorems respectively, are evaluated.**

Keywords: **Database Management Systems, Relational Databases, NoSQL Databases, ACID, CAP, BASE**

I. INTRODUCTION

The advent of computer systems and the rapid changes in industrial dynamics on several fronts including research and technical knowledge increased the demand on quality and productivity of products and services. This saw the automation of real world processes and the introduction of Assembly Automation Equipment, Automated Bookkeeping and Manufacturing systems among a many others. These systems were capable of manipulating only textual and numerical data using Flat file databases as a data management system. This enabled measurement, collection, transcription, validation, organisation, storage, aggregation, update, retrieval and protection of data.

A Flat file database describes any of the various means to encode a database model (most commonly a table) as a single file. Flat file databases contained a logical collection of records with no structured relations which were in plain text or binary file.

Manuscript Received September 15, 2013.

Innocent Mapanga, Department of Computer Engineering Delhi Technological University, Delhi, India.

Prudence Kadebu, Department of Computer Engineering, Delhi Technological University , Delhi, India.

Flat file databases at the time were quite useful as data management requirements were still very limited and simple. With further advances in technology, flat file databases became inadequate as they could not cater for new data types, data security and growth requirements. Also flat file databases contained no information about data and additional knowledge was required to interpret the files. There was no standard way of storing data as well as a standard of communicating to and from the database, hence it created a lot of inefficiencies.

In the 1970s cord came up with the relational theory that led to the development of the relational Database Management Systems (RDBMS) as a solution to the challenges posed by the flat file database system in the earlier years. Storage of data in RDBMS was done using Tables. Standard fields and records are represented as columns (fields) and rows (records) in a table. Their major advantage was the ability to relate and index information. Security was enhanced in RDBMS and they were also able to adapt to considerable growth of data. Structured Query Language, SQL is the programming language used for querying and updating relational databases. For a long time RDBMS has been the preferred technique for data management purposes. However, RDBMS inability to handle modern workloads has given rise to scalability, performance and availability problems with its rigid schema design. Businesses all over the world, including Amazon, Facebook, Twitter, and Google have adopted new ways to store and scale large amounts of data hence the move away from the complexity of SQL based servers to NoSQL database Systems. NoSQL is a class of database management systems that have been designed to cater for situations in which RDBMSs fall short. It is different from the traditional relational databases mainly in that it is schema-less. This makes it suitable to be used for unstructured data. These engines usually provide a query language that provides a subset of what SQL can do, plus some additional features [1]. This paper is organised as follows: section II will look at NoSQL databases overview. Section III focuses on the NoSQL databases categories, Section IV the NoSQL Query Languages followed by Models for structuring NoSQL databases in section V and lastly the conclusion and future works.

II. NOSQL DATABASES

The NoSQL database approach is characterized by flexibility in storage and manipulation of data, improvements in performance and allowing for easier scalability. Many

different types of these NoSQL databases exist, each one suited for different purposes. Examples include MongoDB whose deployments are at foursquare, Disney, bit.ly, sourceforge, CERN, The New York Times, and others. Hadoop (Apache), Cassandra was primarily used by Facebook for their Inbox Search. Afterwards it was open-sourced and now it is an Apache Software Foundation top-level project, being used by Digg, Twitter, Reddit, Rackspace, Cloudkick, Cisco and others. DynamoDB is used by Amazon, Voldemort is used by Amazon, and Neo4J is used by Adobe and Cisco etc. While RDBMS is transaction oriented and based on the ACID principle, NoSQL make use of either CAP or BASE.

Among several capabilities of NoSQL databases are managing large streams of non-relational and unstructured data, fast data access speeds, availability of data even when system is operating in degraded mode due to network partitions. NoSQL databases provide near-endless scalability and great performance for data-intensive use cases. However, with so many different options around, choosing the right NoSQL database for your interactive Web application can be tricky. In general, the most important factors to keep in mind are as follows:

i. *Scalability.* Adopting the Sharding technique can be useful in achieving scale regardless of the database technology in use. Sharding employs horizontal partitioning which is a database design principle in which rows of a database table are held separately .These tables may then be located on a separate database server or physical locations. Scaling quickly, on demand, and without any application changes has become a determinant factor in Web traffic that has on and off surges. Resource contention between servers like disk, memory and CPU is removed. Intelligent parallel processing and maximization of CPU/Memory per database instance can be done.

ii. *Performance.* Interactive applications require very low read and write latencies. Performance is achieved by distributing load across several servers**.** The database must deliver consistently low latencies regardless of load or the size of data. As a rule, the read and write latencies of NoSQL databases are very low because data is shared across all nodes in a cluster while the application's working set is in memory.

iii. *Availability.* Interactive Web applications need a highly available database. If your application is down, you are simply losing money. To ensure high availability, your solution should be able to do online upgrades, easily remove a node for maintenance without affecting the availability of the cluster, handle online operations, such as backups, and provide disaster recovery, if the entire data centre goes down.

iv. *Ease of development.* Relational databases require a rigid schema and, if your application changes, your database schema needs to change as well. In this regard, NoSQL databases offer a number of important advantages that make it possible to alter data structure without affecting your application.

Supporting distributed processing of large-scale data workloads requires adequate processing frameworks likes Apache Hadoop with the MapReduce engine. The emergence of new forms of traffic profiles driven by the Social Web as well as the growing popularity of E-commerce coupled by the ever increasing interconnectedness of the World where Sites are experiencing variations of traffic through-out the year has resulted in massive surges of writes and read traffic in Sites like Twitter, Facebook, Whatsapp in very short time frames hence the need for infrastructure that adapt quickly. Massive upswings on volumes of data movement across the Internet into storage solutions might have traffic becoming a bottleneck. The popularity of agile development methods call for techniques that offer higher scalability and performance so as to keep up with the ever changing technical environment. In-memory database for high update situations, like a website that displays everyone's "last active" time (for chat maybe). If users are performing some activity once every 40 seconds, then it will push RDBMS to limits with about 5000 simultaneous users for instance, what when the numbers multiplies by 10.

III. NOSQL DATABASE CATEGORIES

A. KEY VALUE STORES

Provide a way of storing schema-less data by means of a distributed index for object storage. The key (data-type) will be displayed on the left and the corresponding value (actual data) on the right as shown in the example below.

Key	Value
Comp3_manufa	Dell
Comp20_processor	IntelCore_i5
Comp3_installedMemory	4GB
comp230_systemType	64-BitOS

Figure 1: Key Value Store

Key/Value store is best applicable where write performance is of highest priority since its schema-less structure allows for fast storage of data.

B. COLUMN ORIENTED DATABASES

Provide a data store that resembles relational tables but also adds a dynamic number of attributes to the model. They use keys but they point to multiple tables.

Row Key Columns.....

Comp3	**Brand**	**processor**	**Memory**	**Sys Type**
	Dell	IntelCore_i5	4GB	64BitOS
Comp8	**Brand**	**processor**	**Memory**	**Sys Type**
	Dell	IntelCore2_duo	3GB	32BitOS
Printer42	**Brand**	**Color**	**Type**	
	Hp	White	4in1	

Figure 2: Column Oriented databases

C. DOCUMENT ORIENTED DATABASES

Data is treated as independent objects and their attributes which are stored as separate documents. Each document contains unique information pertaining to a single object. Document stores recognise the structure of the objects stored. Read and writes can be accomplished at once thus making it faster in performance. Schema-less structure gives flexibility in the wake of changing technologies. Documents are described using JSON or XML or derivatives.

Comp3	dell	IntelCore_i5	4GB	64BitOS
Comp8	dell	IntelCore2_duo	3GB	32BitOS
Comp2	Compaq	intel_celeron	2GB	64BitOS

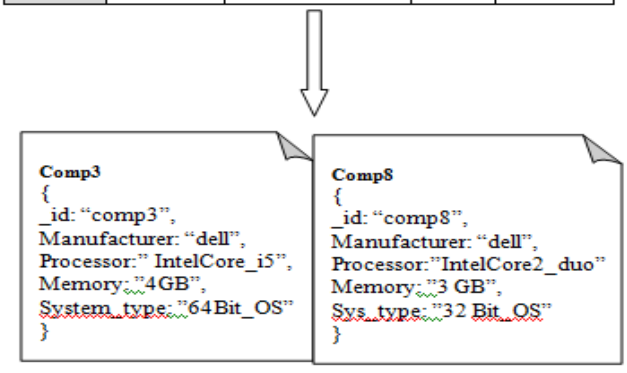
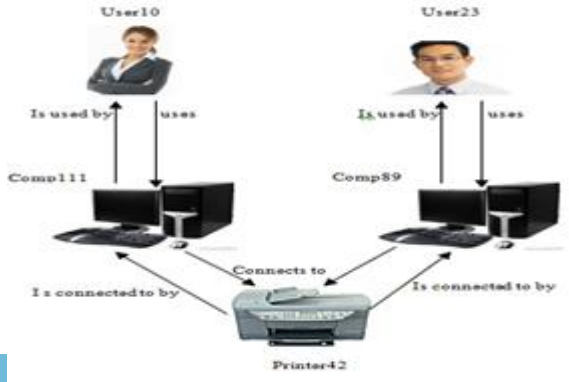


Figure 3: Document Oriented Databases



D. GRAPH DATABASES

These are databases that are based on the graph theory. Graph databases store data in a graph structure with nodes, edges and properties to represent the data. The nodes represent entities in the database. Edges are connecting lines between two nodes representing their relationships. Properties are the attributes of the entities. Graph databases are more applicable in social networks and intelligent agencies as they efficiently show relationships between entities and provide a way to access data in sites with heavy workloads (predominantly reads).

Figure 4: *Graph Databases*

E. OTHER CATEGORIES

The databases discussed above are considered to be the major ones. However, NoSQL has several other categories of databases for various applications. Other types include Multimodel Databases (eg ArangoDB, OrientDB), Object Databases (DB40, Velocity), Grid and C loud Database solutions (Gigaspace, Gemfire), XML Database (BaseX, Berkeley DB XML), Multidementional Databases (SciDB, MiniM DB).

IV. QUERY LANGUAGES

There are a couple of tools that are available for querying numerous NoSQL databases

i. *UnQL(Unstructured Query Language)*

UnQL is an open query language for document databases developed by SQLite and CouchDB teams. It is meant to be a superset of SQL and so in theory it can also be used to access a legacy SQL database engine [9]. However it cannot change its schema. This QL was intended to solve the vendor lock-in problem by providing a cross platform database functionality for document databases like CouchDB, MongoDB and Riak

i. *SPARQL (SPARQL Protocol and RDF Query Language)*

It is a declarative query specification for graph databases designed by W3C RDF Data Access Working Group. It is

	Key Value Stores	Column Family Databases	Document Databases	Graph databases
Based on	Dynamic Hash Tables, Dynamo DB	Google's Bigtable	Lotus Notes, encoding include JSON, XML	Euler's Graph Theory
Data Model	Key/Value pairs	Columns	Key/Value Collections	Graph structure- Nodes, Edges and Properties
Applicability	Handling massive load	Distributed file systems	Web applications, full text searches and updates, information ranking	Semantic web, Social Networks, Intelligent Agencies
Advantages	Simple and easy to implement	Fast querying of data, storage of very large quantities of data	Accepts partially complete data, allows efficient querying	Easy scaling of complex data across distributed systems.
Disadvantages	Inefficient in querying/ updating part of a database	Very low-level API	No standard query language	Traversal of entire graph to give correct results
Examples	Redis,Project Voldermort	Cassandra, HBase	MongoDB, CouchDB	Neo4J, InfoGrid
Data Model	Key/Value pairs	Columns	Key/Value Collections	Graph structure- Nodes, Edges and Properties

Figure 5: Summary of the four categories

able to retrieve and manipulate data stored in Resource Description Framework format [10]. RDF is a directed, labelled graph data format used to represent data in the web [11]. SPARQL specifies four different query variations for different purposes. These include SELECT query, CONSTRUCT query, ASK query and DESCRIBE query

ii. *GQL (Google Query Language)*

It is an SQL like Query Language for retrieving entities or keys from the App Engine scalable data store. Its syntax is similar to SQL.

iii. *SONES Graph Query Language*

The sones GraphQL is a user-friendly domain-specific language and can be thought of as an "SQL for graphs." [16] Sones is an object-orientated graph data storage for a large amount of highly connected semi-structured data in a distributed environment.

iv. *GREMLIN (graph traversal language)*

Gremlin is a domain-specific language hosted in Groovy language which itself is a superset of Java. Gremlin is a graph language. While RDBMS uses JDBC and SQL, graph databases use Blueprints and Gremlin. Gremlin is a style of graph traversal that can be natively used in various JVM languages. Gremlin works over those graph databases that implement the Blueprints property graph data model. Examples include TinkerGraph, Neo4j, OrientDB, DEX, Rexster, and Sail RDF Stores.

V. MODELS FOR STRUCTURING DATABASES

NoSQL emerged as companies, such as Amazon, Google, LinkedIn and Twitter struggled to deal with unprecedented data levels and operation volumes under tight latency constraints. Analyzing high-volume, real time data, such as web-site click streams, provides significant business advantage by harnessing unstructured and semi-structured data sources to create more business value. Traditional relational databases were not up to the task, so enterprises built upon a decade of research on distributed hash tables (DHTs) and either conventional relational database systems or embedded key/value stores, such as Oracle's Berkeley DB, to develop highly available, distributed key-value stores.

A. ACID

The idea of ACID was first coined in the 1970s by Jim Gray [1]. It is a concept that all databases sought to achieve as a way to assure reliability in database systems. A transaction is a transformation of state which has the properties of atomicity (all or nothing), durability (effects survive failures) and consistency (a correct transformation) [5]. The transaction concept emerges with the following properties: Atomicity, Consistency, Isolation and Durability.

ACID transactions provide 4 properties which must be guaranteed:

i. *Atomicity*: A database transaction is treated as a single unit such that all of the operations in the transaction will complete, or none will. This property is referred to as "all or nothing" approach to execution. If one element of the transaction fails, the entire transaction is rolled back.

ii. *Consistency:* This property ensures that there is no violation of integrity thus any transaction will transform the database state from one valid state to another. The transaction must adhere to rules predefined in the system at every instance. If at one instance, a transaction that violates the rules is executed, the transaction is rolled back and the database is returned to the previous valid state. This property entails that there can never be any partially-completed transactions.The database will be in a consistent state when the transaction begins and ends. This property ensures that any transaction will bring the database from one valid state to another. In high availability environment this rule must be satisfied for all nodes in a cluster.

iii. *Isolation:* Every transaction's execution is independent of another and thus will behave as if it is the only operation being performed upon the database. Each transaction has to execute in a "black box" and thus should be transparent to any other concurrent transaction. No transaction should ever see the intermediate product of another transaction until it is completed.

.

iv. *Durability:* After a transaction is committed, the effects thereof are permanent. Any subsequent disturbances or system failure will not result in a change in the current database state.

At every given database operation, all the data undergoes checks to make sure they adhere to constraints imposed by ACID properties. This has worked well for over three decades in normalized, small data environments with less concurrent users in the relational database age. However with new trends in technology and burgeoning internet usage, characterized by Big Data, large number of users and unstructured data in distributed environments which has called for NoSQL databases to cater for the sudden increase in data, invoke a move from ACID properties to CAP.

B. CAP

In the year 2000 Dr Eric Brewer at the ACM Symposium on the Principles of Distributed Computing proposed the CAP theorem. The CAP theorem stated that three essential components namely Consistency, Availability and Partition-Tolerance were crucial for the successful design, implementation and deployment of applications in distributed computing.

i. Consistency: Just as in ACID, Consistency is the property that ensures that all users get the same view of the database at any instance. This enforces adherence to the rules defined in the database.

ii. Availability: Is the property of a database which guarantees that database users always get access to the same version of the database at any point in time.

iii. Partition Tolerance: This property means database can be split over a number of servers such that failure of a single part of the system does not cripple the whole system. The system should be able to operate regardless of undesirable circumstances.

CAP in itself offers a bit of relaxation from the strictness of ACID which may be a bottleneck in some situations (Big Data).

C. PROBLEMS WITH CAP

CAP which is widely adopted as a principle behind the building of distributed systems offers three desirable properties: consistency, availability, and partition tolerance where only two can be chosen and used thereof in these systems. Several flaws were noted:

i. Since one can only choose amongst the three properties, these combinations results in three types of distributed systems: CA (consistent and available, but not tolerant of partitions), CP (consistent and tolerant of network partions, but not available) and AP (available and tolerant of network partitions, but not consistent). CP gives an impression that the system is never available making it a useless system of which it is not the actual case. In using CP, availability is only sacrificed when there is a network partition, meaning that the roles of A and C in CAP becomes asymmetric in practice. It can be seen in this case that the issue of A and C being asymmetric causes a problem.

ii. It very difficult to give the practical differences between CA systems and CP systems. CP systems give up availability only when there is a network partition.CA systems are not tolerant of network partitions meaning they lose availability when there is network partition, making CA and CP identical. This reduces the number of systems to two CP/CA and AP

iii. The lack of latency consideration in CAP is also a cause for concern as to whether the properties trade off can result in the desired distributed system.

If we consider that CAP derived the important properties of Availability and Consistency from ACID then it means losing either one of these would not be desirable. BASE however

takes a more encompassing stance by incorporating both to some extend at any given time. It takes into cognisance the fact that if a network partition occurs there may be partial failures but not complete system failure thus data will still be available amid little delays. BASE stands out as a more robust technique in a NoSQL database than CAP. Dwelling on CAP as a tool for the design of modern scalable databases system might pose a number of problems.

D. BASE

The fundamentals that have guided data management during the past 40 years were based on the transaction model that embraced the ACID principle now viewed as inflexible and too stringent in the light of unstructured data, frequent updates and access to huge amounts of information stored across several data stores. ACID model works well with relational databases. However, does not fare well in very large distributed systems in which availability and performance are of paramount importance. A more flexible model known as BASE was introduced in line with the move towards NoSQL databases as a way to counter the challenges posed by the ACID model. The ACID model falls short in situations characterised by Big Data of unstructured nature and Big Users. In this regard BASE becomes a more effective model where ACID may be a hindrance to database operation. The acronym BASE represents:

i. *Basically Available*: this property ensures that the database is essentially accessible even when a part of the system fails. This is realised by means of a technique known as sharding or partitioning of data across several servers. Data may be replicated across the servers. This results in high availability of the data regardless of possible failures.

ii. *Soft state:* This is the property which enables transactions to proceed even though updates may take time to propagate to all data stores owing to system disturbances or failure. Inconsistencies are tolerated to a certain extend but the end result will be eventual consistency. Consistency control is relegated to the application layer as opposed to the database layer as in ACID. A refresh will result in update of data otherwise the data becomes stale.

iii. *Eventually consistent:* BASE relaxes the requirement of strict consistency at the end of every operation and only guarantees that the data stores will come to a consistent state at a later stage.

	Flat File Database	**RDBMS**	**NoSQL**
Data Model	Flat File	Tables	Columns, Graph, Document, Key/Value
Schema	Schema-less	Fixed Schema	Schema-less
Query Languages	CQL	SQL	API calls, JavaScript and REST
Integrity Model	None	ACID	CAP, BASE
Applicability	Any	Relational and transactional data	Non-relational data
Security	No security	Limited security mechanisms, vulnerable to SQL injection	Authorisation and authentication weaknesses, no encryption, Multiple interfaces increase attack surface.
Advantages	Simpler to use, Less expensive, suited for small scale use	Ensures data integrity between transactions, better security, supports medium to larger sized organisations, provides backup and recovery controls	Can cater for Big Data, unstructured data and distributed systems
Disadvantages	No support for multi-user access, redundancy and integrity problems	Expensive and difficult to manage in distributed systems, Complex and difficult to learn, not suitable for unstructured data	Security is a concern (no encryption), lack of standard query language, Too many varied databases thus no single solution for different purposes
Examples	MsDOS	Oracle, Postgres, MySQL, Microsoft SQL Server	MongoDB, Cassandra, Neo4J

Figure 6: Summary of flat file database, RDBMS and NoSQL

VI. CONCLUSIONS

The underlying features of the main database management systems namely the Flat File Database, RDBMS and NoSQL were reviewed. The main problems found on the Flat file and RDBMS that were common to both database systems include security vulnerabilities, scalability limitations, and availability of data regardless of network partition, timely propagation of changes to ensure consistency, performance bottlenecks and existence of a single point of failure. Owing to the rigid schema of the RDBMS, not all data structures can be represented and stored. These challenges manifest as a result of the architectural constraints inherent in the databases. It was observed that these DBMS have some aspects that are still desirable for instance to achieve reliability and integrity. Completely doing away with the traditional databases in favour of total adoption of the NoSQL also poses great challenges in our data management quest. NoSQL has challenges of not adequately catering for relational and transactional data. While giving cognisance to mission critical data, transactional data and a varied more cases where we seek to ensure reliability as a key aspect, NoSQL may not be ideal, calling for a revisit to the good old mature, tried and tested RDBMS. Owing to this scenario, both RDBMS and NoSQL are suited for different purposes and therefore cannot be absolute substitutes for each other.

Having gone into an analysis of CAP, widely talked about as a tool behind the design of modern scalable database system, we find it falling short in providing suitable engineering tradeoffs in building scalable databases. The lack of latency consideration in CAP significantly reduces its overrating as a preferred choice behind the building of NoSQL databases. This leaves one with important questions for CAP implementation such as: How then does the system make a trade-off between availability and consistency in the event of a partition (P)?

The only feasible solution in the future for a single universal solution to cater for both relational and non-relational data would be an extension on the RDBMS to allow it to cater for non-relational data. Our future works will be exploration of a solution which can assimilate the desirable features of the RDBMS and those of the NoSQL solutions in one database model.

REFERENCES

[1] Alexandru Boicea, Florin Radulescu, Laura Ioana Agapin, "MongoDB vs Oracle - database comparison", IEEE 2012

[2] Kris Zyp http://www.sitepen.com/blog/2010/05/11/ nosql-architecture/, May 2010

[3] http://www.rackspace.com/blog/nosql-ecosystem/

[4] Ruxandra Burtica, Eleonora Maria Mocanu, Mugurel Ionuț Andreica, Nicolae Țăpuș, "Practical application and evaluation of no-SQL databases in Cloud Computing", IEEE 2012

[5] Jim Gray, "The Transaction Concept: Virtues and Limitations", Proceedings of Seventh International Conference on Very Large Databases, June 1981

[6] Vibneiro,http://ivoroshilin.com/2012/12/13/brewers-cap-theorem-explained-base-versus-acid/,December 2012

[7] Anders Karlsson, http://karlssonondatabases.blogspot.com/ August 2013

[8] http://datastax.com/docs/1.0/ddl/column_family

[9] http://www.infoq.com/news/2011/08/UnQL

[10] http://www.wikipedia.org/wiki/SPARQL

[11] W3C,http://www.w3.org/TR/rdf-sparql-query,March 2013

[12] Vibneiro,http://ivoroshilin.com/2012/12/13/brewers-cap-theorem explained-base-versus-acid/,December 2012

[13] Dmitriy kalyada,http://blog.altoros.com/four-things-to-consider-when-choosing-a-db-for-your-interactive- application.html,June 11,2013

[14] Charles Roe, http://www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing/ March 2013

[15] Mike Chapple, http://databases.about.com/od/ other databases/a/Abandoning-Acid-In-Favor-Of-Base.htm, August 2013

[16] Sones GmbH http://en.wikipedia.org/wiki/Sones_GraphDB May 2011

Innocent Mapanga. Innocent Mapanga is currently pursuing Mtech in Computer Engineering at Delhi Technological University in India. The author has completed Bsc (Honors) in Computer Science from Bindura University, Zimbabwe. He is presently working on many papers. His other areas of research include Mobile and Adhoc Networks, Security, Algorithms, Artificial Intelligence, Genetic Algorithms, Databases and Distributed Computing.

Prudence Kadebu. Prudence Kadebu is currently pursuing Mtech in Software Engineering at Delhi Technological University in India. The author has completed Bsc (Honors) in Computer Science from Midlands State University, Zimbabwe. She is presently working on many papers. Her other research areas include Security, Requirements Engineering, Software Quality Management, Virtual Reality, Artificial Intelligence, Distributed Computing, Databases and Data Mining.