Big data Notes

1. “Big data is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation
2. Volume is data “Size”
3. To read a 1TB with read speed of disk as 100MB/sec (HDD) Shall take = 2^40/(100*60*60) = approx. 3 hours
4. Variety also includes variability in terms of structure if structured!
5. Variety- It can be – “structured” – un-structured (text/image/video), – semi-structured (mix)
6. Velocity here refers to: – Speed (Amount of data per unit time) that is to be taken up for processing
7. Often data needs to be loaded into some database system before they can be efficiently processed. This operation is referred to as “data ingestion”
8. Traditional databases work on “fixed schema”. It has two problems

- Putting semi-structure data into fixed schema is not feasible
- Data Records with variable structure also can not be fitted into fixed schema
- It is fixed schema that makes data loading slow in traditional databases!

9. The following are major concerns while using relational databases for big data*
10. Do not scale horizontally – are not designed to run on clusters (Volume challenge)
11. Fixed Schema (Variety Challenge)
12. “Data loading” is slow – load time validation of schema and other constraints makes loading slow (Velocity Challenge)
13. File Processing Systems: Hadoop, Spark, Pig, Hive, Spark-SQL
14. Advantages: – We can directly process simple files like CSV, TSV, JSON; even can run SQL like queries – No loading of data is required

in a DBMS kind of system – Need not to define any schema before processing the file. While running structured query schema validation can happen at query run-time. – Computation can be massively parallelized

15. Disadvantages: – “Only sequential processing” and “Slow” as we do not index or so – file can not be processed in part, and many more…!
16. DBMS type systems – Storage is much efficient and managed by a system – Indexing and efficient access paths are possible – Query Execution can be optimized and run efficiently
17. Disadvantages – Require defining schema (though many no-sql dbms do come with flexible/no schema) – Require loading the data before processing which can be forbidding for adhoc queries
18. Any Big Data Process System should support these to some degree!
19. Scalability (“Elastic Scalability”) • Availability (Reliably) • Consistency
20. Elasticity is the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible
21. Vertical Scaling is expensive, requires data migration, and will always have some limits. • Have no Elastic behavior.
22. Relational (SQL) databases find hard to scale this way!(Horizontally)(“Too much normalization and requiring joins are the main bottlenecks)

Distributed File System (on Cluster)DFS

1. A file system abstraction over a “cluster of computers”
2. A DFS typically provides – Transparency – systems can be used as a stand-alone while data file is “partitioned” and “replicated” – Availability, Scalability, Fault Tolerant, Consistency
3. Few examples of DFS systems are -

4. Google file system- The system is built from many inexpensive commodity components that often fail. – Detecting and recovering is a regular activity
   -Read Workload: large streaming read. Advance steadily through the file rather than go back and forth. • Write Workload: large sequential writes – appends • Concurrent write support only for appends
5. GFS Uses Master-Slave Architecture • One Node is a coordinator Node (Master Node) and others are data nodes (Chunk Servers)
6. Chunk Servers – Data files are stored in parts (splits of files) on chunk servers – Typically each chunk is 16-64MB – Each chunk is replicated (usually 2x or 3x) • Master Node – It does all the coordination work – Stores metadata about files (mapping to chunk servers, access rights, etc.) – It itself can be replicated
7. A data file is "partitioned" in "data chunks" and spread over multiple data nodes in a cluster
8. Each "data chunk" is replicated on multiple data nodes by "replication factor"
9. snapshot and record append operations. Snapshot creates a copy of a file or a directory tree at a low cost. Record append allows multiple clients to append data

- Map Reduce

1. Users specify a _map_ function that processes a key/value pair to generate a set of intermediate key/value pairs
2. a _reduce_ function that merges all intermediate values associated with the same intermediate key.
3. The "map" and "reduce" functions execute in parallel.
4. The MR system invokes M "map" tasks and R "reduce" tasks for a MR job. R is user input where as M system decides.
5. The MR system – Selects node for map tasks, typically the one that has data – "mapper" – "map" function is shipped to mappers – Paradigm shift "computation moves to the data" not the other way round!

6. Inputs and Output to map and reduce functions are always in key-value pair
7. Once all map tasks are over, outputs of mappers are “shuffled” to reducers by “partitioning mapper outputs” based on their key.
8. Partitioning is typically done by applying a simple hash function “hash(k) mod R”
9. Execution happens for each file split in parallel; called mappers – Typically MR master nodes invoke M map tasks, typically one for each split
10. If k is the key of mapper output then partition is done by applying a hash function ‘Hash(k) mod R’
11. Data actually move here for computation from “mappers” to “reducers”
12. Selection condition can be a map only job.
13. Master keeps metadata like List of Map tasks and Reduce Tasks with their state of execution States are: “idle”, “in-progress”, or “completed” ; For each completed map task, the master stores the locations and sizes of the R intermediate file regions produced by the map task.
14. Completed” map tasks on failed nodes are resubmitted to other nodes. Their output is on the local disk of failed nodes
15. For example, URL is they and we want all URLs of a host to be landed on the same reducer – For this, we may have a partitioning function as “hash(Hostname(urlkey)) mod R”
16. Combiner function can act as a reducer when the reduce function is commutative
17. No schema defined upfront! We have already seen systems like Spark-SQL where Schema is not defined at “data loading” time – Loading here refers to data getting put into data files that to be processed – Such systems have “read time schema validation”. Validation here mean validation of data with its schema!
18. data being structured is independent of Schema definition.
19.

mapper_init(self) #executed once for every mapper, and before mapper

mapper(self, key, value) #map function

mapper_final(self) #executed once for every mapper, and after mapper

reducer(self, key, values) #reduce function

combiner(self, key, values) #combine function

reducer_init, reducer_final, combiner_init, combiner_final

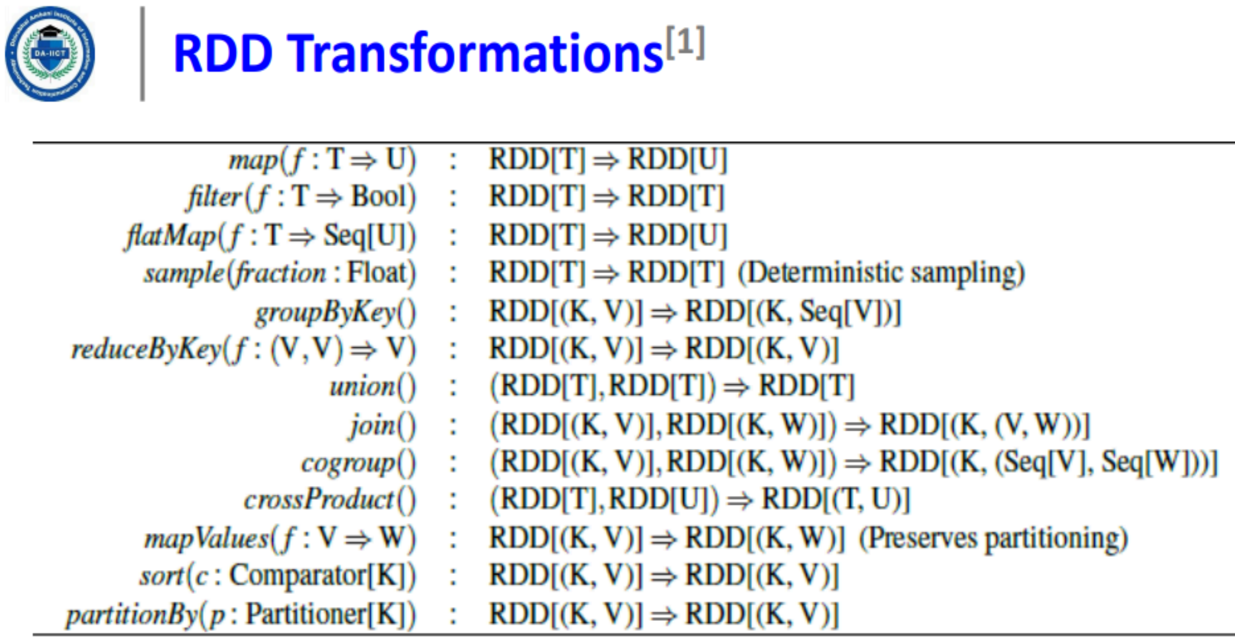
20. MR2-28 pg imp****
21. All the types of join their coding and ask what join will this be?

SPARK

1. Requirement of FULL SCAN of the File – Very inefficient when queries “low selectivity” are to be executed – We can not early terminate the scan. Conditional termination of file processing is not possible.
2. Lack of iteration: If we require iterating through a dataset for multiple times, then every time we read data from disk files; and that happens to be the case with many Analytical and most Machine Learning tasks
3. Lack of Caching: Multiple MR jobs are processing same data almost at the same time. Cashing can help it to run faster x100 times
4. A Unified Engine for Big Data Processing
5. Spark has been created in Scala; and Scala is more native kind of language for Spark
6. “In Memory”, “Distributed”, “Fault Tolerant” collections of objects (called as RDD)
7. Resilient Distributed Dataset (RDD) are collection of distributed, fault tolerant “object collection” partitioned across a set of machines. (Note that “Object collection” here is “in memory”) basically “Distributed, Fault Tolerant List”

8. For Fault Tolerance; RDD objects themselves are not replicated, but maintain information that a partition can be rebuilt if a node fails
9. RDDs are immutable i.e. read only

10. Pair rdd is a collection of key value pair and aggregation operation like “reduceByKey”, “groupByKey” can be applied only on these.
11. Partitions are the unit of parallelism in spark and computations on rdds are performed on partitions
12. Map transformation rdd 1 pg 11
13. Condition for operation is defined by lambda function



RDD Transformations[1]

map(f : T ⇒ U)	:	RDD[T] ⇒ RDD[U]
filter(f : T ⇒ Bool)	:	RDD[T] ⇒ RDD[T]
flatMap(f : T ⇒ Seq[U])	:	RDD[T] ⇒ RDD[U]
sample(*fraction* : Float)	:	RDD[T] ⇒ RDD[T] (Deterministic sampling)
groupByKey()	:	RDD[(K, V)] ⇒ RDD[(K, Seq[V])]
reduceByKey(f : (V,V) ⇒ V)	:	RDD[(K, V)] ⇒ RDD[(K, V)]
union()	:	(RDD[T],RDD[T]) ⇒ RDD[T]
join()	:	(RDD[(K, V)],RDD[(K, W)]) ⇒ RDD[(K, (V, W))]
cogroup()	:	(RDD[(K, V)],RDD[(K, W)]) ⇒ RDD[(K, (Seq[V], Seq[W]))]
crossProduct()	:	(RDD[T],RDD[U]) ⇒ RDD[(T, U)]
mapValues(f : V ⇒ W)	:	RDD[(K, V)] ⇒ RDD[(K, W)] (Preserves partitioning)
sort(c : Comparator[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
partitionBy(p : Partitioner[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]

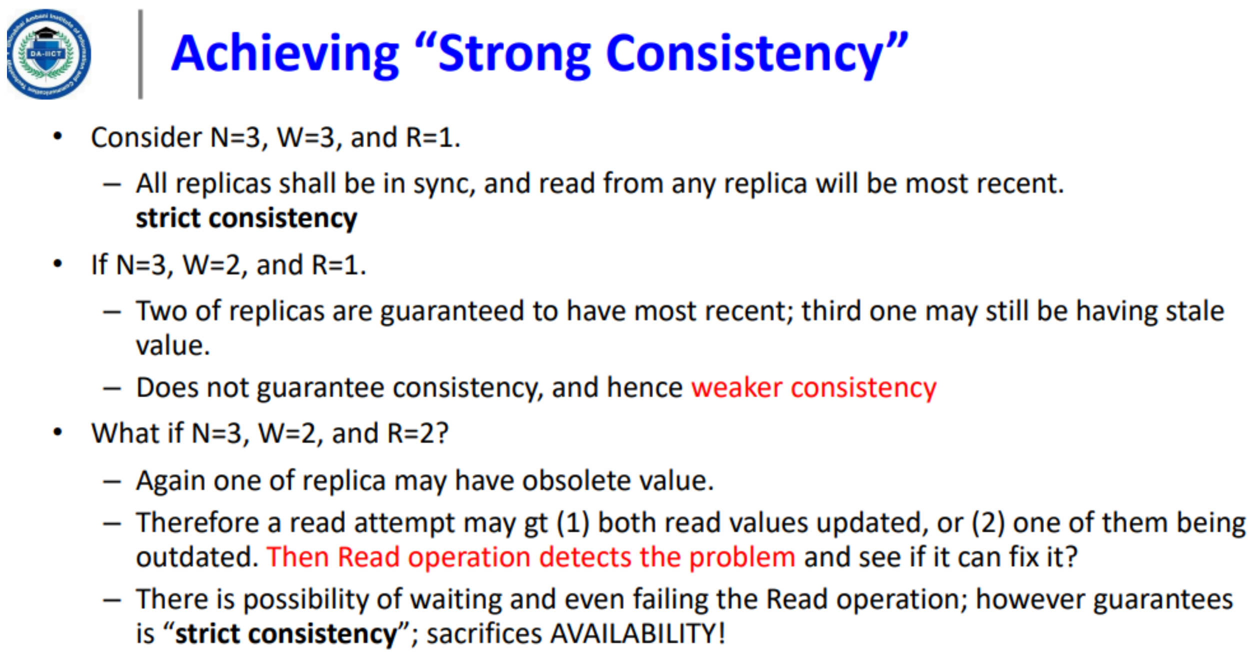
14. So, the approach Spark uses is “do not materialize” the RDD till the time results are required.
15. A “lineage graph” is maintained and is used for materializing RDD data. All operations specified on RDD are added to the lineage graph.
16. RDD transformations are “lazy operations”, i.e. they are evaluated on request of some “Action”
17. Broadcast variables: arrangement of making some data always data available on (data) partitions. Broadcast variables are in Read Only Mode.

18. Accumulators: a global short of data that tasks can accumulate some counters or aggregators.
19. Accumulators typically allow, mappers to accumulate data (aggregated values) in parallel!

No sql databases

1. “Sharding” is a special type of “partition” where data records are partitioned “horizontally”
2. Column friendly or wide column databases, we define column family rather than a specific column.
3. Column Family: A group of columns. A table only has a fixed set of column families but not a fixed set of columns. A column family can have any number of columns and names are not known upfront!
4. While defining a table – Row Key is Required – Column Family is Required – There can be any number of columns in Column Family • A row required to have a “Row Key” and can have any number of column in any column family • A row need not to have columns in all column families!
5. Benefits of aggregation oriented databases, it becomes easier to partition/ distribute data typically based on aggregate key. Computation gets localized.
6. Aggregation can be referred as reverse of Normalization.
7. Embedding- aggregation , referencing - normalization
8. Note that when we do normalization in “Aggregation Oriented” databases • We require splitting a “order object” into two “persistent entities”
9. Most No SQL database systems are aggregation oriented except graph databases
10. Aggregate oriented databases are biased towards a “query load”
11. There are two techniques that are used to ensure ACID – “Serializability” through “two-phase locking” or “timestamp ordering” – “Recovery” from partially-done operations/transactions

12. Consistency needs to be redefined in this context of “distributed and replicated” databases!
13. when replicas are supported, one would want the replicas to always have consistent states.
14. Here Eric identifies three desirable properties in a “transaction processing system”: Consistency (C), Availability (A), and Partition Tolerance (P), and hence CAP • The theorem states that a system can have at most two of these three! This means a system can be either CA systems, or CP systems, or AP systems
15. Another term “BASE” is also around, that is the acronym for Basically Available, Softstate, Eventual consistency, was coined by Brewer et al. in 2000 [1] (in the same talk in which they proposed CAP theorem)
16. Causal consistency: If process A has communicated to process B that it has updated a data item, a subsequent access by process B will get the updated value.
17. Read-your-writes consistency is an important model where process A, after having updated a data item, always accesses the updated value and never sees an older value
18. It should easy to appreciate that if $R+W > N$ (where N is replication factor), then “Strong consistency” can be achieved. R is the number of replicas for read and w is the number of replicas it has to write to pertain a successful write.

19. 

Achieving "Strong Consistency"

- Consider N=3, W=3, and R=1.
  - All replicas shall be in sync, and read from any replica will be most recent. **strict consistency**
- If N=3, W=2, and R=1.
  - Two of replicas are guaranteed to have most recent; third one may still be having stale value.
  - Does not guarantee consistency, and hence weaker consistency
- What if N=3, W=2, and R=2?
  - Again one of replica may have obsolete value.
  - Therefore a read attempt may gt (1) both read values updated, or (2) one of them being outdated. Then Read operation detects the problem and see if it can fix it?
  - There is possibility of waiting and even failing the Read operation; however guarantees is "**strict consistency**"; sacrifices AVAILABILITY!

20. A Highly Read available system may have R=1 and W=N • A highly Write available system may have W=1 and R=N • Or compromise by having W+R < N. In this case systems provides eventual consistency rather than strict consistency

Olap

1. Analytical queries often referred as "Online Analytical Processing (OLAP)" Queries
2. OLAP queries are run on data warehouses.Some of the features of OLAP systems are as following – Optimized for read operations – Keeps the "pre-computed data summaries" stored in the form of multidimensional scheme. – Data Structure is often un-normalized! – Efficiently processing of multi-dimensional data – Need not to support ACID
3. Data Warehouse- "A decision support database that is maintained separately from the organization's operational database
4. For example: Order-No, Order Date, Payment Details, etc in an ecommerce business are not included in the "sales data warehouse" of an e-commerce application.

5. When data is moved to the warehouse, data are brought on to a “single schema”!
6. A physically separate store of data transformed from the operational environment
7. Requires only two operations in data accessing: – Initial loading of data and – Reading of data(Non volatile data warehouse)
8. OLTP. Acronym for “Online Transaction Processing”
9. A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management’s decision-making process.
10. “Materialized View” data warehouse
11. Data cubes- Typically Cubes uses “Star Schema” that contains A “Fact Table” stores actual “summary data” for multiple dimensions – Dimension Tables. That stores more information about the dimension
12. Data cube to snowflake schema when data is normalized
13. Pivot: reorient the cube, visualization, 3D to series of 2D planes – drill across: involving (across) more than one fact table – drill through: through the bottom level of the cube to its back-end relational tables (using SQL)
14. Node: Cuboid – that contains “aggregated values” (called as measures) for “a dimension(attribute) combination”
15. child node has one (exactly one) extra attribute (dimension) than its parent – child node cuboid can be computed from the parent cuboid aggregations
16. Rolap use relational database and bit mapped indices
17. “Multidimensional data cubes are stored on disks in specialized multidimensional structures

In query optimization

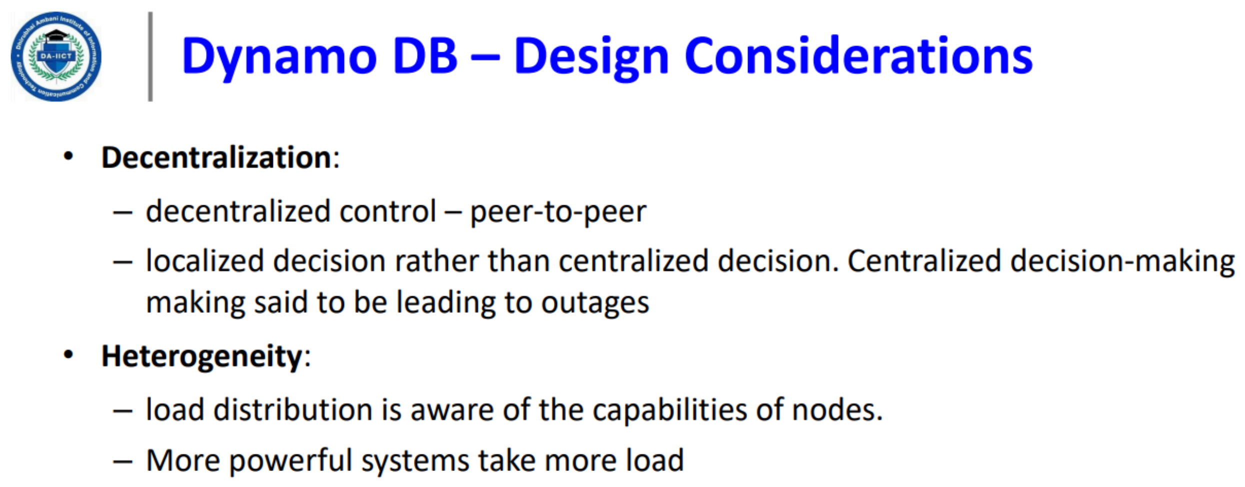
1. All the statements are cached as Abstract Syntax Tree (AST)
2. Data Frames are built on top of the RDD and are “structured”, that is have schema attached with them.

3. We can call them “Schema aware RDDs” - element type is “Row with Schema”
4. For each table, Spark stores its metadata in metawarehouse
5. Unmanaged: – Unmanaged tables have “storage location” specified. – Spark only creates the metadata for these in the meta-store not in metawarehouse.
6. Distributed by cannot be used with order by or clusters by
7. User defined function- UDF

Dynamo Db

1. Created as a “Highly Scalable”, and “Highly Available” system as a service at AWS. It compromises “Consistency” for availability, and supports eventual consistency.
2. The state of objects are stored as binary objects and queried only based on the Key .
3. Dynamo targets applications that need to store objects that are relatively small (usually less than 1 MB).
4. ACID Properties: do not require,Amazon’s services can work with Key-Value model and do not need any relational schema
5. The system needs to function on a commodity hardware infrastructure
6. Dynamo is used only by Amazon’s internal services. Therefore security etc. is taken care of somewhere else.
7. Resolve Write Conflicts at Read-Time
8. Doing it at write has the problem of “write failures”
9. When an attempt to overwrite a later version with an older version, then it is not done and is considered as “un-resolved”. In such cases, both versions are presented to the user and let the user correct it.

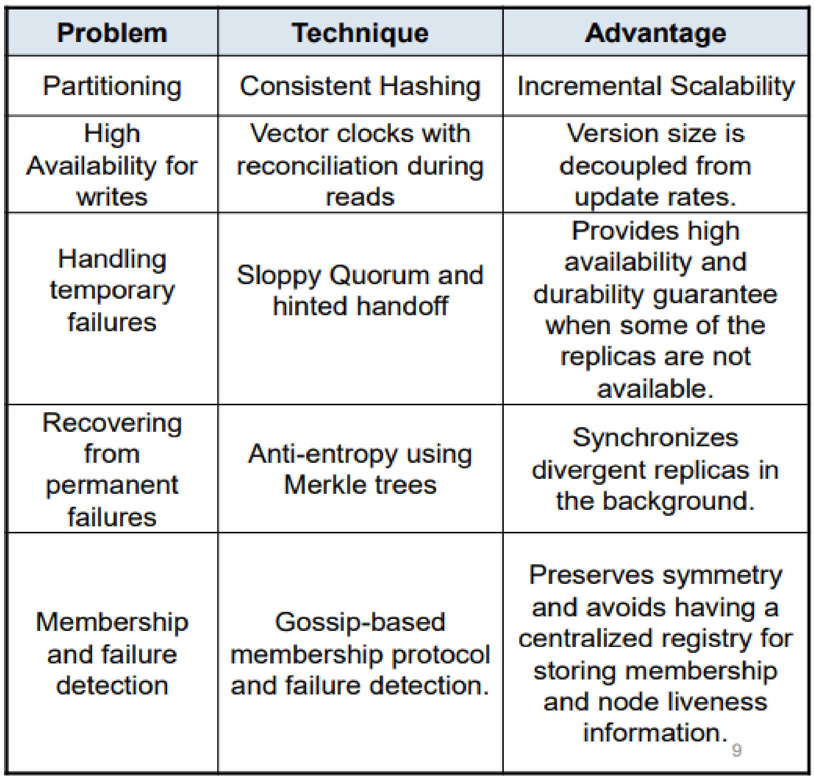
10. Incremental scalability: Dynamo should be able to scale out in one node at a time fashion without bringing the system down
11. Symmetry: All nodes in Dynamo have the same set of responsibilities. Peer-to-Peer System. Any node can be used for Reading and Writing.
12.



Dynamo DB – Design Considerations

- **Decentralization**:
  - decentralized control – peer-to-peer
  - localized decision rather than centralized decision. Centralized decision-making making said to be leading to outages
- **Heterogeneity**:
  - load distribution is aware of the capabilities of nodes.
  - More powerful systems take more load

13. Dynamo Implementation



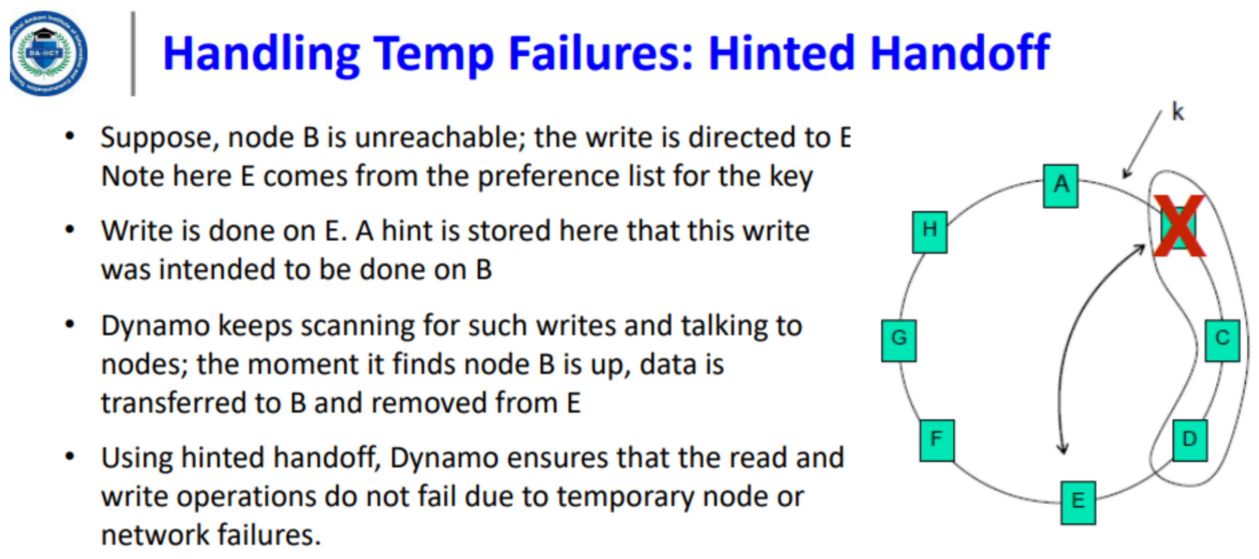
Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

13. Consistent hashing is used for partitioning and replication in dynamo db, it uses vector clocks.

14. “Partitioning” is done based on determining the appropriate node for a search Key
15. “Partitioning” means which data object goes to what node!
16. Suppose we distribute our data on N nodes, and have the following formulation determining node for a data object with key K node = nodeList[hash(K) % N]
17. Solution to network congestion is – creating copies of hot pages to nearer places – called caches – Requests to such pages are diverted to different caches.
18. using Consistent Hashing for – Quickly add copies – Efficiently structure and store the cache – Divert requests to appropriate cache in a “load balanced” manner
19. Thereafter, Consistent Hashing seems to be the standard approach for dynamic partitioning. DHT like Chord and Cassandra are notable examples
20. When a new node is added, there should be a minimum shifting of data and still maintaining uniform distribution. For adding the Nth node, from each older node, only 1/N search keys, on average, should move to the new node. – For example, there are 60 keys distributed on 3 nodes – approx. 20 keys for each node. – If 4th node is added, making N=4; then 1/N of 20 (i.e., 5) would move from each node to the newer node; making 15 keys to each of the 4 nodes. – This typically reduces the movement of 60 nodes to 15.
21. “Consistent Hashing” and suggests solutions for them: First: uniform distribution of data across nodes • Second: It is unfair to assume that all nodes have the same power, more powerful nodes, in terms of RAM, CPUS, Disk, etc., should be able to take more load! • To address these issues, Dynamo DB uses the concept of “virtual nodes”
22. The idea of “virtual nodes” is that we have multiple random representations on the ring for each physical node.

23. Servers A, B, C have multiple representations, randomly placed on the ring - Virtual Nodes • This way, each node gets share from different segments of ring.
24. In terms of loads: More Virtual nodes for more powerful systems. – By having representation of a node in various small-small segment, should be able to get fair distribution of keys over physical nodes
25. The list of nodes that is responsible for storing a particular key is called the preference list. It is to be ensured that the preference list contains more than N physical nodes.Replication is done on the next N-1 physical nodes, where N is the replication factor!(contradicting statements idk)
26. Write always creates a new version of data. Can be performed on any node.
27. Vector clock holds a vector of pairs; where the counter is local for every node! Every data version keeps a record of its writes as "clock vectors"
28. xi is i-th version of x, a typical vector clock for x is: VC(xi)=([s1,3],[s2,1],[s3,2]), Note here server refers first replica that wrote the version
29. Understanding data versioning, pg 4 se in dynamo db2
30. Conflict resolution
31. Get(key) operation locates the object replicas associated with the key in the storage system and returns a single object or a list of objects with conflicting versions along with a context • The put(key, object) operation determines where the replicas of the object should be placed based on the associated key, and writes the replicas to disk.
32. There are two strategies that a client can use to select a node: (1) route its request through a generic load. A request can land to any node, that node in turn forwards request to coordinator node for that key(application code is not tied to partition rings) (2) use a partition-aware client library that routes requests directly to the appropriate coordinator nodes. (lower latency)
33. Common (N,R,W) configuration used by several instances of Dynamo is (3,2,2)

34. Strong consistency can be achieved by having “R+W > N”
35. (3, 3, 1): fast write, slow read, less consistency, not very durable – (3, 1, 3): fast read, slow write (but never fails), higher consistency, durable
36. A node handling a read or write operation is known as the coordinator. Typically, this is the first among the top N nodes in the preference list
37. Read and write operations use first N healthy nodes in the preference list - skipping over down or inaccessible ones
38. request lands to the coordinator node • the coordinator generates the vector clock for the new version and • writes the new version locally. • The coordinator then sends the new version (along with the new vector clock) to the N highest-ranked reachable nodes. If at least W-1 nodes respond then the write is considered successful.
39. request lands to the coordinator node • the coordinator requests all existing versions of data for that key from the N highestranked reachable nodes in the preference list for that key, • and then waits for R responses before returning the result to the client. • If the coordinator ends up gathering multiple versions of the data, it • returns all the versions it deems to be causally unrelated. • The divergent versions are then reconciled and the reconciled version superseding the current versions is written back.
40. Dynamo folk feel that nodes may become un-available temporarily and may come back quickly – A node becomes un-available for a longer duration then it becomes a permanent failure. For that “replica synchronization” keeps working in the background. • For temporary failures, temporary nodes can be used for writing temporarily, and data are sent back to the targeted node. The technique is called as “Hinted Handoff”
41. A node becomes un-available for a longer duration then it becomes a permanent failure. For that “replica synchronization” keeps working in the background
42. For temporary failures, temporary nodes can be used for writing temporarily, and data are sent back to the targeted node. The technique is called as “Hinted Handoff”



Handling Temp Failures: Hinted Handoff

- Suppose, node B is unreachable; the write is directed to E Note here E comes from the preference list for the key
- Write is done on E. A hint is stored here that this write was intended to be done on B
- Dynamo keeps scanning for such writes and talking to nodes; the moment it finds node B is up, data is transferred to B and removed from E
- Using hinted handoff, Dynamo ensures that the read and write operations do not fail due to temporary node or network failures.

43.
44. “Sloppy quorum” because it can not gurantee reading same N set of replicas. • Said to be using “N healthiest nodes” from preference list
45. Hinted handoff works best if the system membership churn is low and node failures are transient.
46. DynamoDB uses Merkle-Tree for comparing replicas!
47. There are scenarios under which hinted replicas become unavailable before they can be returned to the original replica node. To handle this and other threats to durability, Dynamo implements an anti-entropy (replica synchronization) protocol to keep the replicas synchronized.
48. Dynamo could use “vector clocks” for comparing replicas but would take too long. It doesn’t***
49. higher entropy mean “higher inconsistencies” in replicas
50. Dynamo has no central node that keep important information like ring information. All peers keep it. Add and deletion in dynamo doesn’t happen automatically, but manually.
51. Node add request can be given to any node in the ring, that node communicates this “membership change” request to other peer nodes, this change request propagates to other members in “gossip manner” on peer to peer basis.(epidemic protocol)
52. Data items in Dynamo Db are schema less.

53. “Partition-Key(hash key) + Clustering Key(sort key)” should be uniquely identifying a data item in the table!(primary key) When specified both Primary Key said to be “Composite”
54. Partition Key is used for data partition • Clustering or Sort Key is used for ordering data within a partition
55. We can perform GET based on part of Primary Key(hail dynamo db)
56. Secondary index is stored external to table, that is mapping information is external to the table, it could also be partitioned. It basically stores index primary key to table primary key**
57. “Access-Key” of secondary index again has – “Partition Key” , and – Sort Key [optional]
58. A secondary index is created to query the data through “another attribute(s)” • Example: Consider Music Table; Let us have secondary index based on “Gener” as Partition Key, and “Album Title” as sort Key. This means – Index entries are partitioned on “Gener” and sorted within partition based on “Album Title” • On having this secondary index, we can have look-up based on another key “Genre” and “Album Title”
59. “Index keys” are additional “search keys” , it’s values are the record of the items. Dynamo db maintains indexes automatically
60. Describe Table– Returns information about a table, such as its primary key schema, throughput settings, and index information
61. batch‑write‑item – Writes up to 25 items to a table/
62. The batch variations are more efficient than calling single item operation multiple times because, the application only needs a single network round trip to read/write the items.
63. Query: – Can be only on Key – However, we may only specify partition key, sort-key is optional – Filter can be applied on any attribute • Scan is complete scan of a table – Filter can be applied on any attribute.
64. PUT operations (Primary Key based) – Parameters: pair or List of pairs if bulk load – put‑item(K,V), batch‑write‑item (LIST(, Update‑Item (K, update‑instructions)

65. GET operations (Primary Key based) – Parameters: Key (values for Partition Key and Sort Key) or Set of Keys. – Returns: Data Item object, or List of Data Items – get-item (KEY) , batch-get-item (KEY-LIST)
66. QUERY (Partition Key based) – Parameters: Partition Key – Returns: Data Item object, or List of Data Items matching the specified Partition Key value. – Result can be further Filtered
67. SCAN – Scans full table. – Returns: List of all Data Items. – Result can be further Filtered.(ascending and descending possible)
68. For questions like, which user has the highest score in NFS we can use querying using indexing because we have to look in just NFS ka data and not scan through the whole document, however in cases where we have to scan through the entire document we have to do scanning.
69. Global secondary index has different set of partition key and a sort key that can be different from those on the base table, where as • Local secondary index has the same partition key as the base table, but a different sort key(These are local in the sense that they are in the same partition with just a different sort key and cannot go beyond that partition)
70. A typical characteristics of secondary indexes in dynamo-db is that they can have additional attributes
71. We can retrieve items from table through GSI using the Query and Scan operations only. – GetItem operations can't be used on GSI
72. Dynamo db uses eventual consistent read. However if we set consistent read parameter = true then strong consistency is used.
73. A strongly consistent read might not be available if there is a network delay or outage. In this case, Dynamo DB may return a server error (HTTP 500). They are also not supported on global secondary indexes.
74. Join aur vo sab queries kaisey likhte vo dekhna padega.
75. Inserts are best in “sequential inserts” Whereas searches are good when Indexes are used, BUT INSERTS and Updates are not efficient for high-speed “data ingestion”

76. LSM trees was proposed as a solution to high speed data writing while also provisioning support for fast searching.
77. Writes (INSERT/UPDATE/DELETE) are done in main memory in append-only (logging manner) – They are, in turn, flushed to secondary storage in batches. LSM Tree achieves better write efficiency than traditional b+-trees
78. LSM is a multi-level tree. Level 0 is an in-memory tree. Other levels are on disk. Writes are always in Level-0 (in a LOGging manner). When Level-0 hits its limits, it is flushed to disk and merged into a Level-1 tree. Reads require looking into trees at all levels starting from level 0 to the last level. For pointed queries, we can stop wherever found but for range searches, we may have to go up to the last level.
79. The process of merging higher-level trees to lower level is referred to as "compaction"
80. Modern systems said to maintaining access pointers to reach to right sets of "structures"
81. For bloom filters, no is a no but yes maybe a no
82. When we merge a tombstone with a tombstone, just keep one copy. When we merge a tombstone with a key-value pair, we delete the key-value pair
83. Log-structured": – All data is written sequentially (in a "logging" manner), regardless of logical ordering
84. Results are collected from multiple trees in a "sort-merge" manner – Periodic Merger also happens on trees on disk
85. Compaction affects latency
86. Two types of common compactions are found in the literature – Tiered LSM Trees, and – Levelled LSM Trees, based on greediness of compaction
87. If compaction is done at the same time of "flushing" to the next level then it is levelled .
88. Tiered is more "write" optimized as less amount of work at the time of writing • Levelled is more "read optimized" as less numbers of "runs" for reads
89. SSTables are immutable

90. All writes go directly to the MemTable. When full, the MemTable is flushed to disk as an SSTable. Periodically, on-disk SSTables are collapsed together (Compaction). Reads check the MemTable first and then the SSTable indexes
91. Leveldb is “Tree” at each level in the original LSM is replaced by a set of SSTables. In memory sstable is called memtable. To ensure recovery, entries are first logged, in line with “WAL”
92. Note here Level-0 here is in both: Memory and Disk. baki sab samw
93. SSTable contains a sequence of blocks; typically size of 64KB
94. SSTable stores “block index is at the end of the SSTable file.
95. The index here is simply a list of “row-key -> block address” sorted on the key
96. Bigdata table is distributed, sparse, distributed, persistent, multidimensional sorted map.
97. For webtable, rowkey is URL, column family, column name;
98. In this case we have three column families: Content, language, anchor(href), a value is used as column-name - “cnnsi.com” as column. only latest 3 versions of data are kept.
99. Operations - put get delete scan
100. Data of an entire column family can be deleted but for a column cannot be deleted.
101. 64 bit integer are encoded as an 8 byte big endian value
102. Bigtable uses GFS to store log and data files
103. A tablet(storage unit for bigtable) is of size 100MB to 200MB
104. Each row(transactional unit) is uniquely identified by a rowkey(64kb)
105. The design of Row-Key could be crucial in achieving “data locality” for accessing together. For instance, in “web-table”, pages in the same domain are grouped into contiguous rows by “reversing the hostname components” of the URLs. for example: “maps.google.com/index.html” -> “com.google.maps/index.html”
106. A serving node typically stores 100 tablets
107. A table is “sharded” into blocks of contiguous rows, called tablets start-key and end-key.

108. The benefits of having a large number of smaller tablets per machine are the following: – Fine-grained load balancing – Fast recovery: say 100 machines each pick up 1 tablet from a failed machine
109. A tablet is built of multiple SS Tables. Tablets do not overlap (for row-key ranges”), SSTables can overlap
110. The big table system has three major components – Client library – Tablet Servers (many, handle read write; splits tablet when they get too big) – Master Server (single) – Chubby cell
111. Client library: available at each client – It maintains the soft state of index: (key range) -> (table server location) mappings.
112. A chubby cell – ensures there is a single master – stores bootstrap location of bigtable data – discovery and death finalization of tablet servers – store bigtable schema and access control information
113. Bigtable uses three level hierarchy, first is the file stored in chubby that contains the location of the root tablet.
114. The root tablet contains the locations of all of the tablets of a special METADATA table. Each METADATA tablet contains the location of a set of user tablets. User tables typically store “references to SSTables”(SSTables are on GFS) ; root table never splits.
115. Big Table client library is designed to cache metadata information.
116. Before writing to “memTable” an entry is made in tablet’s commit log
117. Important to note here is that “writes” are only in memTable
118. A tablet having t “SSTable stack” takes O(t) SSTable reads to test for a “key”
119. The solution here is “shared logs”, that is one log file per tablet server instead of per tablet. Writes for many tablets co-mingled in the same file!
120. The solution here is “shared logs”, that is one log file per tablet server instead of per tablet – Writes for many tablets co-mingled in the same file!

121. We can have as many columns as you need in the table - can have millions of columns in a table! As long as the row size is below 250MB, beyond 100 is when it causes trouble.
122. In cases where a row goes higher than 100MB, design key such that it comes in adjacent rows
123. row key must be 4 KB or less
124. Big data programming and schema design(pg 23)
125. the term “Column Oriented Databases” refers to the Physical storage of the data
126. Late materialization: i.e. late “tuple reconstruction” refers to delaying the joining of columns into wider tuples. Block-oriented and vectorized processing: the processing is done on blocks of “column vectors” than tuples at a time
127. For example, for sorted column gender column, if first 42 elements of a column contain the value ‘M’, then “Run-length Encoding” compression compresses it to (‘M’, 1, 42) that is
128. “adaptive indexing” is becoming a realistic architectural feature in modern columnstore
129. Physical model in Monet, where all tables have exactly two columns
130. Dropped pure “BAT” concept, and allows storing multiple columns in a “column table”; still relational tables are particularly partitioned for storage
131. Dropped MIL layer so that the relational operators can process (vectors of) multiple columns at the same time
132. Each column-file contains data from one column, compressed using a columnspecific compression method, and sorted according to some attribute in the table that the column belongs to. This collection of files is known as the “read optimized store” (ROS)
133. Additionally, newly loaded data is stored in a write-optimized store (“WOS”), where data is uncompressed and not vertically partitioned.
134. Periodically, data is moved from the WOS into the ROS via a background “tuple mover” process, which sorts, compresses, and writes re-organized data to disk in a columnar form

135. Each column in C-Store may be stored several times in several different sort orders. Groups of columns sorted on the same attribute are referred to as “projections”