Distributed Databases and Principles of Disaster Recovery

Nicholas Schmidt, CIPP/US

iapp

Distributed Databases and Principles of Disaster Recovery

When it comes to databases, privacy professionals care most about design decisions and systems architecture, as well as how databases are used and maintained, and less about the individual variables and other minutia involved in creating them.

In this second part in a series of white papers for technology pros, we'll look at some of the major facets of modern database design and management with a focus on the large internet-based database systems that define modern technology, including distributed databases, errors and backups.

Distributed databases

Traditionally, databases were located on one machine (system/computer) and either accessed directly through that machine or remotely by connecting to it via the internet. When many users needed to access the same database, it was usually placed on a mainframe, a very large and powerful computer. This strategy has some advantages, including ease of maintenance and greater database consistency, since there is only one system to maintain.

After the turn of the millennium, parallel processing (the use of multiple computer processors units aka "processor cores" to run a single program) became more common as traditional mainframes began to be replaced by smaller, individually less powerful computers called servers. The term "server" is short for "web server," because these computers are most commonly used for hosting websites. With the shift away from mainframes, large databases also shifted from a centralized model to a distributed one, where the same database was spread across multiple systems in such a way that it was still possible for a user to access the full data of the database. Such distributed databases are everywhere in modern life: Google Drive, Microsoft's OneDrive, Dropbox, and other cloud storage systems are, at brass tacks, distributed databases being offered to the public for document storage purposes. Most modern web applications utilize a distributed database for data and user management. A distributed database is also one of the base components of all blockchain applications.

Distributed databases offer several advantages over the mainframe model. The most significant advantage is faster processing speed when handling large numbers of users. Imagine you are buying groceries with 29 other shoppers, and each person needs five minutes to check out. If the supermarket only has one employee ringing up customers and you're last in line, you'll spend two-and-a-half hours waiting! If there are two employees, the wait time drops to an hour and 15 minutes, and with 10 employees, it becomes 15 minutes. Each server in a distributed database is like an additional supermarket clerk at checkout; by balancing the load between itself and its peers, queries can be taken in parallel, allowing each user to use the site faster.

The redundancy of distributed databases causes them to be both more reliable (the probability that the database will be accessible to a user at any random time) and more available (the probability that the database will stay accessible over a period of time) as system failures will be isolated to only portions of the database, leaving the remainder to continue functioning as normal. This same redundancy also offers logistical advantages, including the ability to swap, repair, update or add machines in a distributed database without taking down the entire system.

There are two broad methods of creating a distributed database. The first is duplication, where the contents of the database are copied from a "master" or parent database and placed onto another system. The second is fragmentation or allocation, where the contents in the parent database are split apart and stored on different systems in a way that still allows the querying of the full database via communication across the network of the database's host systems. These two techniques can be used together, fragmenting certain aspects of the parent and duplicating others, creating a partially replicated distributed database. By contrast, if the parent database is completely copied, the distributed database is fully replicated. Completely fragmenting a database is called non-redundant allocation.

There are advantages and disadvantages to both design methods. Fragmenting increases query speed and decreases computational expense because a query can be quickly routed to the right node(s) in the network and then resolved by only having to search over a subset of the data. The same applies to database updates: The database can be more quickly updated in fragmented sections. By contrast, duplicated databases are more expensive and time intensive to query or update, since a query must search through the entire database to resolve. However, fully replicated databases are more secure and reliable because fraudulent updates and data corruption are more likely to be confined to a single node of the database and be corrected through comparison with redundant nodes. Additionally, because nodes are copies of each other, if one node goes down, no data is lost.

Database errors, faults, recovery and rollback

System failures are inevitable in any computer system, including databases, and cannot be completely prevented. When talking about them, proper vocabulary is important. The Institute of Electrical and Electronics Engineers defines the most common terms as follows:

Failure: Program behavior that is noticeably incorrect to the user.

Fault: A defect in the code or construction of the system that causes a failure, also known as a bug.

Error: A human mistake or action triggering the fault, such as in input, coding, etcetera.

Failures in databases most often result in incomplete database updates, which can then go on to serve as faults triggering additional failures. For example, if a new record is being added to a table with six required fields (which must not be null for the record to function properly), but the database encounters a failure halfway through and only three of the required fields are written, the record will be unusable. Every time the system attempts to read the fourth required field, it will only read null and fail. Depending upon what the missing fields are and how they are used, minor failures in reading them may lead to large failures, perhaps cascading throughout the database and making the entire system unusable.

It is therefore important that incomplete database updates be prevented from subsisting in the database after a failure; this is called recovery. In every case, the database must be restored to the state it was in before the update; the update must be rolled back. If other updates have been made based on the update that caused the failure, it is necessary to roll them back to maintain consistency. This is called a cascading rollback, which is computationally expensive and should be avoided as much as possible. To effectively rollback an update, two things are necessary:

1. A list of changes the update has made to the database in a format that can be used to either redo or undo those changes. This list is called a log or changelog.

2. A point in time where one can conclude that the transaction has been, or will be, successfully complete and doesn't have to be rolled back. This is called the commit point.

Additionally, many recovery algorithms create a temporary location in memory where database updates can be written before being placed into the actual database. Such temporary media in computing are referred to as buffers or caches, depending upon why they are holding the data.

Because a failure could happen at any time, it is important that recovery algorithms record a change they are going to make to the database in the log before making it. This is called write-ahead logging and allows recovery and rollback of all changes in the update after failure using the log.

Backing up a database and failover

Occasionally, a failure or other event causes catastrophic damage that renders the entire database unusable. Examples of such catastrophic failures include destruction of required host servers and widespread corruption of the database. In these circumstances, mere rollback is insufficient; the database must be restored from a complete copy called a backup. When not stored on hard disk, database backups are generally stored on backup tapes, which function like the tape cassettes popular before the turn of the millennium but can store data at an order of magnitude far greater than the average hard drive. In recent years, solutions have increasingly materialized that digitally replicate data at another location without need for physical media, aided by the development of storage area network technology, in which many servers operate as though they are a single hard drive.

A modern backup tape system

It is good practice for every organization to have a regular backup schedule where the current contents of their databases are regularly written to tape. Large companies with dedicated data centers usually create at least one tape backup every night. It is common to store backup tapes in a different location than the servers they were created from to insulate them from destruction in natural disasters. Businesses that do not run their own backup schedules often hire a vendor to perform these services for them. When restoring a system from a backup, the first step is to replace the ruined version of the database with that contained in the backup. The second task is to use any surviving logs to redo the committed updates made after the last backup was taken. This is called roll forward.

In addition to backup, popular and business-critical web services maintain a redundant but unused separate system set up to immediately take over in the event the primary version of the service goes offline. This is called failover, undoing it and restoring the primary system is called failback. Failback is automatic and utilizes a heartbeat, a connection between the two systems where the main system tells the backup system it is operating normally at regular intervals, usually every couple of seconds. If a heartbeat is not received, the backup system will automatically activate and assume the duties of the primary system. Manual failover is called switch over.

Disaster recovery plans

It is important for every organization that regularly intakes and handles data to have a disaster recovery plan detailing how it will be handled and data services restored in the event of a service disruption. Disaster recovery plans are occasionally called Continuity of Operations Plans, especially in the U.S. public sector. In developing a plan, there are three important metrics:

1. The recovery point objective is the maximum amount of time for which data can be lost due to a disaster; if you want to be able to restore your data up to two days before any catastrophic system failure, your RPO is 48 hours. It is important to include the time it will take to receive the necessary materials and restore the system to functioning when calculating RPO: any data submitted to the system during that time will also be lost. The RPO for a company that does nightly backups is not 24 hours; it can be closer to 72 hours.

2. The recovery time objective is the maximum amount of time a system can be down due to a disaster. If your organization has an RTO of 24 hours, a working system should be up and running within that timeframe.

3. The recovery consistency objective is the percentage of data in the database that should be intact and useable after it has been restored from a disaster.

All disaster recovery plans contain three broad types of strategies:

1. Preventative strategies are employed as part of normal business operations to reduce the risk and impact of a disaster. Examples include access control, failover, server backup procedures, and good software development and installation strategies.

2. Detective strategies are employed to detect a disaster and pinpoint its source when possible. Automated methods include heartbeat, as well as agent-based and agentless monitoring systems. Agent-based monitoring systems require the installation of an agent, or piece of code that reports or performs tasks on behalf of another controlling piece of software on another system, on each of the monitored components, including individual servers. "Agentless" systems use methods built into an operating system as their agents. This means that, while agent-based systems are generally more specialized and secure, agentless solutions are easier to administer and (generally) cheaper.

3. Corrective strategies are employed to rectify a disaster. They include operational responses by different business departments, notifications (when required), and management practices like post-incident reviews.

Just like any other emergency plan, it is crucially important to test the disaster recovery plan from time to time.

Test your knowledge

You are the chief information officer of DumpBin, a cloud storage company based in New York City, and you've tasked one of your interns with creating a first pass at a disaster recovery plan. Unfortunately, the plan you got back has some problems. Use what you've learned to identify the issues and suggest solutions.

DUMPBIN DISASTER RECOVERY PLAN

Recovery goals

To allow DumpBin to completely recover in a timely manner, this plan sets the following goals:

RPO: 6 hours
RTO: 24 hours
RCO: 75% [Assume this RCO is correct.]

Preventative strategies:

- Several methods, including strong access control, that will be covered in a future white paper.
- Backup the full database to tape at 10 p.m. every night, when site usage is lowest. Send the tapes to the warehouse in Philadelphia.
- Manual failover of the main portal to an adequate backup, administrated by a worker in the data center.
- Non-redundant allocation of the database to limit the ability of viruses to spread within the system.
- Provisions for a yearly review, test, and revision of this plan.

Detective strategies:

- Use of an agentless monitoring system to monitor several of our specialized internal metrics.
- Use of an agent-based monitoring system to monitor the standard networking metrics of each server in the main database and identify when a node goes down.

- Use of a heartbeat system to monitor individual server uptime and downtime.

- Detailed system logging to catch errors and faults, including write-ahead logging to allow for the analysis of database transactions.

Corrective strategies:

- If the failure was a data breach, contact legal and follow their instructions regarding notifications and preservation of information.

- List and order of people across the business who need to be notified in the case of various disasters together with their contact information.

- List of relevant DumpBin outside counsels with their contact information and instructions for when they need to be contacted.

- List of relevant outside vendors and their functions and contact information.

- If necessary to restore the SAN from a backup, bring the most recent backup tape from Philadelphia by car (one-and-a-half hours) and upload it into the database (four hours).

- Detailed technical instructions for responding to different scenarios, including acts of God.

- Location of pre-created outage messages to place on website in different circumstances.

Answers:
The errors are as follows:

1. The RTO and RPO are flipped. The RTO refers to the maximum amount of time the system should be down due to catastrophic failure while the RPO refers to the maximum amount of time for which system data can be lost. We can tell that the two numbers are flipped because a six-hour RPO is impossible given DumpBin's nightly backup schedule but is barely feasible given the time frame needed for restoration.

2. A 24-hour RPO is not feasible because it doesn't take the time necessary to restore the back-up into account, which will matter in the worst case. To see this, assume there is a catastrophic system failure at 9:59 p.m. that prevents the nightly backup from being taken. Because the backup could not be taken and (assuming the RTO is accurate as is, see the next point) it will take six hours to restore the system. Therefore, the RPO should be at least 30 hours, not 24. The intern's mistake is probably because he correctly noticed that failover will allow users to continue using DumpBin without noticeable disruption and therefore limit any data loss to that data which has changed in the 24 hours before the last back-up and that that information can be transferred into the main system after it is restored. However, when calculating RPO, we must think about the worst case and that means assuming failover will fail.

3. A six-hour RTO may not be feasible either. Although it is true that a backup tape can reach the main data center in an hour and a half and can be completely uploaded in four, and it is theoretically possible that IT can fix whatever problem or reset the system in the hour and a half it takes the backup tape to arrive, this is probably not a realistic estimate. We know from the plan that DumpBin is a SAN, which likely incorporates many servers. Fully resetting a system, including manual reinstall of the DumpBin host software, is probably not feasible in the two hours the team have to meet the deadline given the four-hour upload time, especially if we assume that the failure happened at a time the IT team is mostly out of the office, like late at night. See the next two points in the next block for other reasons why this estimate is inaccurate.

4. There is no provision in the plan to roll-forward the restored database from any surviving logs. Therefore, the restoration from backup will be incomplete, and DumpBin will throw out some recoverable data. The RTO also does not consider the time needed for roll-forward.

5. There is no provision or conditions for failback after the disaster has been fixed. Time necessary to failback will need to be included in RTO and clear criteria for a “ready” system should also be included.

6. Manual failover isn’t called failover but switchover. However, it should be noted that, for a cloud offering like DumpBin, perceived reliability and availability is everything. Therefore, DumpBin should consider investing in an automatic/true failover system.

7. Agentless and agent-based monitoring are switched here. The agentless monitoring system cannot create custom metrics because the embedded agents do very little processing and are often deployed automatically. The agent-based monitoring system can be used to monitor standard networking metrics like ping, but an agentless monitoring system would be more effective for this task.

8. Write-ahead logging is wrongly placed in detective measures when it should be a preventive measure. Remember the main advantage of write-ahead logging is recovery from future failures, which is a preventive rather than detective task.

9. A non-redundantly allocated database does not provide any more security from viruses than a redundantly allocated one. This is because the actual software managing the database is the same, only the data contained in each node is different. However, replication in a database can help prevent data corruption and fraudulent updates; therefore, some replication may be desirable in DumpBin’s database, although, given its nature as a cloud storage app, full replication probably isn’t feasible. This error is tricky. Don’t feel bad if you didn’t spot it.