Back-End Optimisation

COMP6205: Web Development

Dr A. Rezazadeh
December 16
Introduction

• Web applications often need data from other Servers/organisations, and to interact with them
  – hence distributed data access is essential
• There are many technical solutions, none ideal
• Historically, the two main approaches are
  – data-oriented (message or document passing)
  – procedural (RPC, service oriented)
• These are both equally powerful
  – message passing can simulate calls and vice versa
• What matters is performance, reliability, trust, ...
Performance

• The Internet can deliver high bandwidth, but latency is still poor (30 msec or more)
  – at most 30 messages or calls per second
  – around 10 times slower than most LANs

• This gives a significant advantage to coarse grained approaches
  – fewer, larger, interactions
  – transfer the whole object, not a single attribute
Distributed Database Interactions

- We rule out solutions which rely on synchronous calls
  - RPC (Remote Procedure Call) or synchronous service invocation
- And also database techniques such as
  - direct ODBC connection to external data, or distributed transactions
  - these work well over a LAN, within one organisation
  - but not over the internet, and between organisations
- Distributed transactions use two phase commit (2PC)
  - all parties signal agreement to the transaction coordinator
  - who then orders them to make the required updates
  - all affected data is locked until the transaction completes
  - which is a problem if there are any network delays or crashes
RDBMS Properties

• **Transaction** - Basic unit of work in an RDBMS

• **ACID**
  
  – Atomicity: all or nothing
  
  – Consistency: always leave a database in a consistent state
  
  – Isolation: every transaction is completely isolated
  
  – Durability: once committed always recorded
Distributed DB Systems - CAP Theorem

- **CAP Theorem (Brewer’s Theorem)** state: it is impossible for a *distributed* Database system to simultaneously provide all three of the following guarantees:
  - **Consistency**: all nodes see the same data at the same time
  - **Availability**: Node failures do not prevent other survivors from continuing to operate (a guarantee that every request receives a response about whether it succeeded or failed)
  - **Partition tolerance**: the system continues to operate despite arbitrary partitioning due to network failures (e.g., message loss)

- A distributed system can satisfy only part of these guarantees at the same time but not all three.
CAP Theorem

• Conjectured by Prof. Eric Brewer at PODC (Principle of Distributed Computing) 2000 keynote talk

• Described the *trade-offs involved in distributed system*

• It is impossible for a web service to provide following *three guarantees at the same time*:
  • Consistency
  • Availability
  • Partition-tolerance
A popular misconception: 2 out 3

• Of the following three guarantees potentially offered by a distributed systems:
  – Consistency
  – Availability
  – Partition tolerance

• Pick two

• This suggests there are three kinds of distributed systems:
  • CP AP CA

Any problems?
CAP Theorem Revisited -12 year later

• Prof. Eric Brewer: father of CAP theorem

  – “The “2 of 3” formulation was always misleading because it tended to oversimplify the tensions among properties. ...

  – CAP prohibits only a tiny part of the design space: perfect availability and consistency in the presence of partitions, which are rare.”

http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed
Consistency or Availability

• Consistency and Availability is not “binary” decision
  – AP systems relax consistency in favor of availability – but are not inconsistent
  – CP systems sacrifice availability for consistency - but are not unavailable
  – This suggests both AP and CP systems can offer a degree of consistency, and availability, as well as partition tolerance

• In Web applications failures can, and will, occur to the networked system
  – Therefore partitioned tolerance should be accommodated.
CAP Theorem Implications

• **AP**: A partitioned node returns
  – a correct value, if in a consistent state;
  – a timeout error or an error, otherwise
  – e.g., DynamoDB, CouchDB, and Cassandra

• **CP**: A partitioned node returns the most recent version of the data, which could be stale.
  – e.g., MongoDB, Redis, AppFabric Caching, and MemcacheDB
CAP Theorem: demonstration

• A simple proof using two nodes:
CAP Theorem: demonstration

- A simple proof using two nodes:

A

B

Not Consistent!

Respond to client
CAP Theorem: demonstration

- A simple proof using two nodes:

Wait to be updated

Not Available!
CAP Theorem: demonstration

- A simple proof using two nodes:

Not Partition Tolerant!

A gets updated from B
Why this is important?

- The future of databases is **distributed** (Big Data Trend, etc.)

- CAP theorem describes the **trade-offs** involved in distributed systems.

- A proper understanding of CAP theorem is essential to **making decisions** about the future of distributed database **design**.

- Misunderstanding can lead to **erroneous or inappropriate** design choices.
Problem for Relational Database to Scale

• The Relational Database is built on the principle of ACID (Atomicity, Consistency, Isolation, Durability)

• It implies that a truly distributed relational database should have availability, consistency and partition tolerance.

• Which unfortunately is impossible ...

• So in reality, there are only two types of systems ... i.e., if there is a partition, does the system give up availability or consistency?
Types of Consistency

• **Strong Consistency**
  – After the update completes, **any subsequent access** will return the **same** updated value.

• **Weak Consistency**
  – It is **not guaranteed** that subsequent accesses will return the updated value.

• **Eventual Consistency**
  – Specific form of weak consistency
  – It is guaranteed that if **no new updates** are made to an object, **eventually** all accesses will return the last updated value (e.g., *propagate updates to replicas in a lazy fashion*)
Eventual Consistency - A Facebook Example

- Bob finds an interesting story and shares with Alice by posting on her Facebook wall
- Bob asks Alice to check it out
- Alice logs in her account, checks her Facebook wall but finds:

  - Nothing is there!
Eventual Consistency - A Facebook Example

- Bob tells Alice to wait a bit and check out later
- Alice waits for a minute or so and checks back:
  - She finds the story Bob shared with her!
Eventual Consistency - A Facebook Example

• Reason: it is possible because Facebook uses an eventual consistent model

• Why Facebook chooses eventual consistent model over the strong consistent one?
  – Facebook has more than 1 billion active users
  – It is non-trivial to efficiently and reliably store the huge amount of data generated at any given time
  – Eventual consistent model offers the option to reduce the load and improve availability.
Dynamic Tradeoff between C and A

• An airline reservation system:
  – When most of seats are available: it is ok to rely on somewhat out-of-date data, **availability is more critical**
  – When the plane is close to be filled: it needs more accurate data to ensure the plane is not overbooked, **consistency is more critical**

• Neither strong consistency nor guaranteed availability, but it may significantly increase the tolerance of network disruption.
Discussion

• In an e-commercial system (e.g., Amazon, e-Bay, etc), what are the trade-offs between consistency and availability you can think of? What is your strategy?

• Hint -> Things you might want to consider:
  – Different types of data (e.g., shopping cart, billing, product, etc.)
  – Different types of operations (e.g., query, purchase, etc.)
  – Different types of services (e.g., distributed lock, DNS, etc.)
  – Different groups of users (e.g., users in different geographic areas, etc.)
ACID vs. BASE

• **BASE**: (Basically Available, Soft-State, Eventually Consistent)

• **Basic Availability**: fulfill request, even in partial consistency.

• **Soft State**: abandon the consistency requirements of the ACID model pretty much completely

• **Eventual Consistency**: at some point in the future, data will converge to a consistent state; delayed consistency, as opposed to immediate consistency of the ACID properties.
  
  – purely a *liveness* guarantee (reads eventually return the requested value); but
  
  – does not make *safety* guarantees,
  
  – an eventually consistent system can return any value before it converges.
BASE and the Rise of NoSQL Databases

• **SQL databases:**
  – Structured query language for Traditional relational databases (unique keys, single valued, no update/insertion/deletion anomalies)
  – Well structured data
  – ACID properties should hold

• **NoSQL databases:**
  – triggered by the storage needs of companies such as Facebook, Google and Amazon.
  – Not necessarily well structured – e.g., pictures, documents, web page description, video clips, etc.
  – Lately of increasing importance due to big data.
  – ACID properties may not hold.
  – focuses on **availability** of data even in the presence of multiple failures.
  – spread data across many storage systems with a high degree of replication.
Scalability – Scale-up

- **Vertical Scaling (Scale-up):** Generally refers to adding more processors and RAM, buying a more expensive and robust server.

- **Pros**
  - Less power consumption than running multiple servers
  - Cooling costs are less than scaling horizontally
  - Generally less challenging to implement
  - Less licensing costs
  - (sometimes) uses less network hardware than scaling horizontally
Scalability – Scale-up

• **Vertical Scaling (Scale-up):** Generally refers to adding more processors and RAM, buying a more expensive and robust server.

• **Cons**
  – PRICE, PRICE, PRICE
  – Greater risk of hardware failure causing bigger outages
  – generally severe vendor lock-in and limited upgradeability in the future
Scalability – Scale-out

**Horizontal Scaling (Scale-out):** Generally refers to adding more servers with less processors and RAM.

This is usually cheaper overall and can literally scale infinitely (although we know that there are usually limits imposed by software or other attributes of an environment’s infrastructure).

**Pros**

- Much cheaper than scaling vertically
- Easier to achieve fault-tolerance
- Easy to upgrade
Scalability – Scale-out

- **Horizontal Scaling (Scale-out):** Generally refers to adding more servers with less processors and RAM.

- **Cons**
  - More licensing fees
  - Bigger footprint in the Data Center
  - Higher utility cost (Electricity and cooling)
  - Possible need for more networking equipment (switches/routers)
Parallelism

• Modern servers are multi-threaded of course

• But true parallelism requires replicated hardware

• Some workloads are easy to parallelise
  – in the banking system, for example, each user can only access their own account details
  – so users can easily be partitioned across multiple app servers and database servers

• In such a case, N replicated servers can increase throughput of that tier by a factor defined by Amdahl's Law
Amdahl's Law

• Gene Amdahl, chief architect of IBM's first mainframe series and founder of Amdahl Corporation and other companies found that:

  – there were some fairly stringent restrictions on how much of a speedup one could get for a given parallelized task. These observations were wrapped up in Amdahl's Law:

• If F is the fraction of a calculation that is sequential, and (1-F) is the fraction that can be parallelized, then the maximum speed-up that can be achieved by using P processors is $1/(F+(1-F)/P)$. 

COMP6205
Amdahl's Law – Examples

- If 90% of a calculation can be parallelized (i.e. 10% is sequential) then the maximum speed-up which can be achieved on 5 processors is $1/(0.1+(1-0.1)/5)$ or roughly 3.6 (i.e. the program can theoretically run 3.6 times faster on five processors than on one).

- If 90% of a calculation can be parallelized then the maximum speed-up on 1000 processors is $1/(0.1+(1-0.1)/1000)$ or 9.9 (i.e. throwing an absurd amount of hardware at the calculation results in a maximum theoretical (i.e. actual results will be worse) speed-up of 9.9 vs a single processor).
Amdahl's Law – Observations

• The point that Amdahl was trying to make was that using lots of parallel processors was not a viable way of achieving the sort of speed-ups that people were looking for.
  – i.e. it was essentially an argument in support of investing effort in making single processor systems run faster.

• Amdahl's Law is a statement of the maximum theoretical speed-up you can ever hope to achieve.
  – The actual speed-ups are always less than the speed-up predicted by Amdahl's Law
Amdahl's Law – Observations

• Why actual speed ups are always less?
  – distributing work to the parallel processors and collecting the results back together is extra work required in the parallel version which isn't required in the serial version.

• Anyone interested in parallel solutions simply MUST have a very deep understanding of Amdahl's Law if they are to avoid having unrealistic expectations of what parallelizing a program/algorithm can achieve and if they are to avoid underestimating the effort required to achieve their performance expectations.
Web servers

Apache

- Home Page: http://httpd.apache.org/
- Cost: Open source, free, no licensing fees
- Advantages:
  - Is flexible because of ability to pick and choose various modules
  - Has enhanced security (notable, because vulnerabilities typically are attacked in Windows-based machines)
  - Has strong user-community support
    
    Runs on UNIX, Windows, Linux, Mac OS
- Disadvantage:
  - Is a process-based server, which means each simultaneous connection requires a thread that can incur significant overhead
Microsoft IIS

- **Home Page:** [http://www.iis.net/](http://www.iis.net/)
- **Cost:** Comes with Windows (could mean increased costs through licensing)
- **Advantages:**
  - Is supported by Microsoft
  - Provides access to .NET framework & ASPX scripts
  - Integrates with other Microsoft services (Active Directory, MS SQL server, ASP, etc.)
- **Disadvantage:**
  - Isn’t able to customize as much as open-source web servers
- **Popular Sites Using It:** (mainly Microsoft properties) Live.com, Bing.com, Microsoft, MSN, Barbie.com
NGINX

• Home Page: [http://wiki.nginx.org/Main](http://wiki.nginx.org/Main)
• Cost: Free (open source)
• Advantages:
  – Is known for speed and for being a reverse-proxy server
  – Is an event-based server, which enables high performance and scale
  – Is potentially better for a VPS (Virtual Private Server) environment
• Disadvantages:
  – It is not as feature-rich as the other two
• Popular Sites Using It: Wordpress.com, Pinterest, Tumblr, Instagram, CNet, Meetup.com, Weibo, Time.com
Others

• **LiteSpeed**
  • Advantages:
    – Runs at high speed
    – Is compatible with Apache
    – Popular Sites Using It: (No high-profile sites, see)

• **Google Web Server (GWS)**
  • Home Page: [https://developers.google.com/cloud/](https://developers.google.com/cloud/) (Google Cloud)
  • Popular Sites Using It: Google sites (various), Blogger
Monitoring Web Server Performance

• Web servers tend to experience performance issues when Web application traffic increases unexpectedly.

• Some common issues that cause Web servers to perform poorly include:
  – memory consumption,
  – unresponsive services,
  – disabled Web plugins and permissions,
  – and heavy server loads.
Benefits of monitoring Web server performance

• Get real performance data about your Web server health and end-user experience.

• Receive proactive alerts on Web server performance

• Delivers both historical and current web server performance metrics, delivering insight into the performance over a period of time.

• Performance graphs and reports are available instantly. Reports can be grouped and displayed based on availability, and connection time.
Monitoring Solutions

• Nagios
  – Check services, hosts for outage
  – Highly configurable, extendable
  – Worth your time investment

• Ganglia

  • Ganglia is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids. It is based on a hierarchical design targeted at federations of clusters.
  – Monitor for performance