Big Distributed Software Systems

- Overview of modelling of non-functional properties of distributed systems, with emphasis on issues related to ‘big’ systems, such as scalability, asynchronous archs.

- Many similar concerns to previous system:
  - Many non-functional properties are interdependent and interrelated
  - Tradeoff of non-functional properties
  - Each application (and part of an application) has its own particular cost function, with respect to which trade-offs should be evaluated
  - Financial cost of improving non-functional properties

Fallacies of Distributed Systems

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous

(due to Joy, Lyon, Deutsch, Gosling at Sun)

Non-functional Properties of Distributed Systems

General property of dependability can include many properties, main ones:

- Dependability
  - Reliability
  - Availability
  - Safety
  - Security

Non-functional Properties continued.

- Performance
- Latency
- Transparency
- Scalability
- Flexibility
- Survivability

Reliability, Availability

Availability: fraction of time the system is usable

(sometimes rolled in together with reliability

Techniques:

- Use of redundancy
- Use of fault tolerance techniques

Issues of:

- Consistency
- Security

General performance issues

- Responsiveness

e.g. interactive applications

requiring fast and consistent response

- Throughput
  - rate at which computational work completed
- Quality of service
  - ability to meet the servive requirements of users

- Basic techniques such as of load balancing

Big system performance issues

Memory bottleneck:

Techniqes:

- Memcaching
- Partioning of database into shards

(memcache: distributed memory caching system originally developed for LiveJournal)

Standard architecture model: mySQL + memcache

Newer NoSQL models

Typical configuration

- Master database
  - handles data write operations
- Replication on multiple slaves
  - handle read operations
- Master server synchronizes constantly with slaves
  - If master fails, a slave can become master

Performance enhanced by

- Caching results from database memcache
- Partitioning database into shards

General performance issues

Replication needed for performance

Facebook early architecture issues (Sobel)

70milliseconds to send packet coast to coast USA

Master west coast, replicated slave on east coast

Write to master vs old value in east coast cache

1 to 20 secs delay in synchronisation

… solved!

General performance issues

Databases: basic functions: Create, Read, Update, Delete (CRUD)

Traditional ACID model

- Atomicity : full transaction succeeds or not
- Consistency: database integrity preserved
- Isolation/independence of transactions
- Durability: committed transaction persists

ACID: pessimistic, forces consistency at end of every operation,

General performance issues

Databases:

Newer BASE model

- Basically Available
- Soft state
- Eventually consistent.

Also addressing issue of scalability

Latency

- Latency vs bandwidth
- Rommer(2005)
  - Bandwidth improvement 100 times latency improvement
  - Ultimate Limits:

ping USA to Europe: 30milliseconds

  - Goals:

make as few calls as possible

move max amt data in each call

Latency

- See Pritchett Ebay archietcure Latency discussion

More non-functional properties

Transparency

- Location
- Migration
- Replication
- Concurrency
- Parallelism

Complete transparency not always achievable or desirable

Flexibility

- easier to change … resources, locations, number of users, functionality
- workload balancing

Scalability

Scalability – resilience to cope with inceasing user load

- number of users
- transactions …

Scalability mechanism

- resources
  - computational
  - memory
  - Communication

Rearchitecting

Scalability

Some issues:

- Cost of resources
- Performance loss
- Bottlenecks

General alternative solns:

Scaling up: buying servers with more memory, computing power, with current architecture

Scaling out: altering architecture horizontally and using sharding, caching, replicating servers etc

Performance and Scalability (Barish)

Scalability and performance different

Performance: relates to raw speed of the application, such as seen by one user

For a fixed number N of users, can focus on improving average response time per user

Scalability: relates to accommodating increasing numbers of users.

A “slow” application might scale, a “fast” application might not scale

Performance and Scalability (Barish)

Improving performance involves increasing end-to-end performance.

Measure speed of various parts of query round-trip: client-to-server, server-to-database, database response … etc

Improvement in performance:

Speedup = Tnew/Told

Performance and Scalability (Barish)

Scalability improvement based on metrics from:

Throughput – rate at which transactions are processed by the system

Resource usage – usage levels for the various resources(CPU, memory, disk, bandwidth)

Cost – price per transaction

Performance and Scalability (Barish)

Article at www.informit.com

Various issues discussed, emphasis:

Scalability and performance different

A “slow” application might scale, a “fast” application might not scale

Shared data system model

C: Strong consistency: all clients get a consistent view, even in the presence of data writes

A: High availability: all clients can get a copy of data, even in presence of some replica failures

P: Partition tolerance: integrity of system properties preserved when system is partitioned

No set of failures less than total network failure is allowed to cause the system to respond incorrectly (Gilbert & Lynch)

Brewer's CAP Theorem

- "When designing distributed web services, there are three properties that are commonly desired: consistency, availability, and partition tolerance. It is impossible to achieve all three" Gilbert and Lynch
- Asserted by Eric Brewer 2000, later given a proof by Gilbert and Lynch (as defined by them)

Consistency and Availability

Provided all nodes communicating, can have consistency and availability

Enforce using two-phase commit protocol, cache coherence protocols

(strong guarantees but at a performance hit, only easy for single site, clustered databases)

Consistency and Partition tolerance

Pessimistic locking of distributed databases

Protocols of majority/quorum voting to define agreed consistent data

Can enforce consistency in the presence of temporary partitions at the cost of system-wide blocking

Availability and Partition tolerance

Sacrificing consistency, makes implementation easier and allows for improved performance

Typical web service based applications where optimistic updating, and lack of consistency are resolved eventually, using various mechanisms, for example, using time stamps, transaction identifiers, invalidation protocols

Consistency, Availability, and Partition tolerance

Common Design Trade-off

The system is distributed on an open uncontrolled internet, so will have to tolerate partition

Sacrifice consistency, for availability

Amazon claim that just an extra one tenth of a second on their response times will cost them 1% in sales. Google said they noticed that just a half a second increase in latency caused traffic to drop by a fifth. (Julian Browne 2009)

Brewer's CAP Theorem

- Keynote address by Eric Brewer 2000
- Contrasts newer systems with traditional distributed systems with distributed objects, tight coupling with RPC (remote procedure call), strong ACID guarantees
- Several ideas discussed in the talk
- Traditional emphasis on computation rather than data is wrong

Brewer's Talk

- Traditional emphasis on computation rather than data is wrong
- Achieving persistent consistent data is difficult
- CAP theorem
- Issue of boundaries, need to move away from effort at transparent RPC-like interfaces

(related to the synchronous vs asynchronous coupling discussions)

Brewer's Talk

- Asserts the DQ Principle

  Data/query * Queries/sec = constant = DQ

Fault can affect capacity (Q) i.e. queries per sec

or completeness (D) i.e. data per query, or both

Also asserts: Harvest * Yield ~ constant

i.e. trade-off one against other for given cost

Harvest: fraction of complete result (data)

Yield: fraction of answered queries

Brewer's CAP Theorem

- Succinct statement of certain trade-offs
- As Brewer point out, no property is absolute and there is a basic statistical aspect to all of them
- Principle that encapsulates a useful model of non-functional properties and their trade-off